

The zref-clever package implementation*

Gustavo Barros[†]

2021-09-29

Contents

| | | |
|----------|----------------------------|-----------|
| 1 | Initial setup | 2 |
| 2 | Dependencies | 2 |
| 3 | zref setup | 3 |
| 4 | Plumbing | 7 |
| 4.1 | Messages | 7 |
| 4.2 | Reference format | 8 |
| 4.3 | Languages | 10 |
| 4.4 | Dictionaries | 11 |
| 4.5 | Options | 17 |
| 5 | Configuration | 29 |
| 5.1 | \zcsetup | 29 |
| 5.2 | \zcRefTypeSetup | 29 |
| 5.3 | \zcLanguageSetup | 31 |
| 6 | User interface | 33 |
| 6.1 | \zceref | 33 |
| 6.2 | \zcpageref | 35 |
| 7 | Sorting | 35 |
| 8 | Typesetting | 43 |
| 9 | Compatibility | 68 |
| 9.1 | \appendix | 68 |
| 9.2 | memoir class | 69 |
| 9.3 | listings package | 70 |
| 9.4 | enumitem package | 71 |

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

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

| | | |
|--------------|---------------------|-----------|
| 10 | Dictionaries | 71 |
| 10.1 | English | 71 |
| 10.2 | German | 75 |
| 10.3 | French | 78 |
| 10.4 | Portuguese | 82 |
| 10.5 | Spanish | 85 |
| Index | | 89 |

1 Initial setup

Start the DocStrip guards.

```
1 <*package>
```

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

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

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

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

Identify the package.

```
14 \ProvidesExplPackage {zref-clever} {2021-09-29} {0.1.0-alpha}
15 {Clever LaTeX cross-references based on zref}
```

2 Dependencies

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

```
16 \RequirePackage { zref-base }
17 \RequirePackage { zref-user }
18 \RequirePackage { zref-abspage }
19 \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` property is provided by `zref-base`, while `zref-abspage` provides the `abspage` property which gives us a safe and easy way to sort labels for page references.

The `counter` property, in most cases, will be just the kernel’s `\@currentcounter`, set by `\refstepcounter`. However, not everywhere is it assured that `\@currentcounter` gets updated as it should, so we need to have some means to manually tell `zref-clever` what the current counter actually is. This is done with the `currentcounter` option, and stored in `\l__zrefclever_current_counter_tl`, whose default is `\@currentcounter`.

```
20 \zref@newprop { zc@counter } { \l__zrefclever_current_counter_tl }
21 \zref@addprop \ZREF@mainlist { zc@counter }
```

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.

```
22 \zref@newprop { zc@thecnt }
23 { \use:c { the \l__zrefclever_current_counter_tl } }
24 \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`.

```
25 \zref@newprop { zc@type }
26 {
27   \exp_args:NNe \prop_if_in:NnTF \l__zrefclever_counter_type_prop
28     \l__zrefclever_current_counter_tl
29     {
30       \exp_args:NNe \prop_item:Nn \l__zrefclever_counter_type_prop
31         { \l__zrefclever_current_counter_tl }
32     }
33   { \l__zrefclever_current_counter_tl }
34 }
35 \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’).

```
36 \zref@newprop { zc@cntval } [0]
37 { \int_use:c { c@ \l__zrefclever_current_counter_tl } }
```

```

38 \zref@addprop \ZREF@mainlist { zc@cntval }
39 \zref@newprop* { zc@pgval } [0] { \int_use:c { c@page } }
40 \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”, “super-counter”, “parent counter” 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 `\c1@<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 `\c1@<counter>`, looking for the counter for which we are trying to set a label (`\l__zrefclever_current_counter_tl`, by default `\@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 `\c1@<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.

Recursively generate a *sequence* of “enclosing counters” and values, for a given $\langle counter \rangle$ 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 {\langle counter \rangle}
    \__zrefclever_get_enclosing_counters_value:n {\langle counter \rangle}

41 \cs_new:Npn \__zrefclever_get_enclosing_counters:n #1
42 {
43   \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
44   {
45     { \__zrefclever_counter_reset_by:n {#1} }
46     \__zrefclever_get_enclosing_counters:e
47     { \__zrefclever_counter_reset_by:n {#1} }
48   }
49 }
50 \cs_new:Npn \__zrefclever_get_enclosing_counters_value:n #1
51 {
52   \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
53   {
54     { \int_use:c { c@ \__zrefclever_counter_reset_by:n {#1} } }
55     \__zrefclever_get_enclosing_counters_value:e
56     { \__zrefclever_counter_reset_by:n {#1} }
57   }
58 }

```

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’).

```

59 \cs_generate_variant:Nn \__zrefclever_get_enclosing_counters:n { e }
60 \cs_generate_variant:Nn \__zrefclever_get_enclosing_counters_value:n { e }

```

(End definition for `__zrefclever_get_enclosing_counters:n` and `__zrefclever_get_enclosing_counters_value:n`.)

Auxiliary function for `__zrefclever_get_enclosing_counters:n` and `__zrefclever_get_enclosing_counters_value:n`. They are broken in parts to be able to use the expandable mapping functions. `__zrefclever_counter_reset_by:n` leaves in the stream the “enclosing counter” which resets $\langle counter \rangle$.

```

    \__zrefclever_counter_reset_by:n {\langle counter \rangle}

```

```

61 \cs_new:Npn \__zrefclever_counter_reset_by:n #1
62 {
63   \bool_if:nTF
64     { \prop_if_in_p:Nn \l__zrefclever_counter_resetby_prop {#1} }
65     { \prop_item:Nn \l__zrefclever_counter_resetby_prop {#1} }
66     {
67       \seq_map_tokens:Nn \l__zrefclever_counter_resettters_seq
68       { \__zrefclever_counter_reset_by_aux:nn {#1} }
69     }
70 }
71 \cs_new:Npn \__zrefclever_counter_reset_by_aux:nn #1#2
72 {
73   \cs_if_exist:cT { c@ #2 }
74   {
75     \tl_if_empty:cF { c1@ #2 }
76     {
77       \tl_map_tokens:cn { c1@ #2 }
78       { \__zrefclever_counter_reset_by_auxi:nnn {#2} {#1} }
79     }
80   }
81 }
82 \cs_new:Npn \__zrefclever_counter_reset_by_auxi:nnn #1#2#3
83 {
84   \str_if_eq:nnT {#2} {#3}
85   { \tl_map_break:n { \seq_map_break:n {#1} } }
86 }

```

(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.

```

87 \zref@newprop { zc@enclcnt }
88 { \__zrefclever_get_enclosing_counters:e \l__zrefclever_current_counter_tl }
89 \zref@newprop { zc@enclval }
90 { \__zrefclever_get_enclosing_counters_value:e \l__zrefclever_current_counter_tl }
91 \zref@addprop \ZREF@mainlist { zc@enclcnt }
92 \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}`.

```

93 \tl_new:N \g__zrefclever_page_format_tl
94 \cs_new_protected:Npx \__zrefclever_page_format_aux: { \int_eval:n { 1 } }
95 \AddToHook { shipout / before }
96 {
97   \group_begin:
98   \cs_set_eq:NN \c@page \__zrefclever_page_format_aux:
99   \exp_args:NNx \tl_gset:Nn \g__zrefclever_page_format_tl { \thepage }
100   \group_end:
101 }
102 \zref@newprop* { zc@pgfmt } { \g__zrefclever_page_format_tl }
103 \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

```

104 \msg_new:nnn { zref-clever } { option-not-type-specific }
105 {
106   Option~'#1'~is-not-type-specific~\msg_line_context:~
107   Set~it~in~'\iow_char:N\zcLanguageSetup'~before~first~'type'
108   ~switch~or~as~package~option.
109 }
110 \msg_new:nnn { zref-clever } { option-only-type-specific }
111 {
112   No~type~specified~for~option~'#1'~\msg_line_context:~
113   Set~it~after~'type'~switch~or~in~'\iow_char:N\zcRefTypeSetup'.
114 }
115 \msg_new:nnn { zref-clever } { key-requires-value }
116 { The~'#1'~key~'#2'~requires~a~value~\msg_line_context:. }
117 \msg_new:nnn { zref-clever } { language-declared }
118 { Language~'#1'~is~already~declared.~Nothing~to~do. }
119 \msg_new:nnn { zref-clever } { unknown-language-alias }
120 {
121   Language~'#1'~is~unknown,~cannot~alias~to~it.~See~documentation~for~
122   '\iow_char:N\zcDeclareLanguage'~and~
123   '\iow_char:N\zcDeclareLanguageAlias'.
124 }
125 \msg_new:nnn { zref-clever } { unknown-language-transl }
126 {
127   Language~'#1'~is~unknown,~cannot~declare~translations~to~it.~
128   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
129   '\iow_char:N\zcDeclareLanguageAlias'.
130 }
131 \msg_new:nnn { zref-clever } { unknown-language-opt }
132 {

```

```

133   Language~'#1'~is~unknown~\msg_line_context:..Using~default.~
134   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
135   '\iow_char:N\zcDeclareLanguageAlias'.
136 }
137 \msg_new:nnn { zref-clever } { dict-loaded }
138 { Loaded~'#1'~dictionary. }
139 \msg_new:nnn { zref-clever } { dict-not-available }
140 { Dictionary~for~'#1'~not~available~\msg_line_context:.. }
141 \msg_new:nnn { zref-clever } { unknown-language-load }
142 {
143   Language~'#1'~is~unknown~\msg_line_context:..Unable~to~load~dictionary.~
144   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
145   '\iow_char:N\zcDeclareLanguageAlias'.
146 }
147 \msg_new:nnn { zref-clever } { missing-zref-titleref }
148 {
149   Option~'ref=title'~requested~\msg_line_context:..
150   But~package~'zref-titleref'~is~not~loaded,~falling~back~to~default~'ref'.
151 }
152 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
153 {
154   Option~'hyperref'~only~available~in~the~preamble.~
155   Use~the~starred~version~of~'\iow_char:N\zceref'~instead.
156 }
157 \msg_new:nnn { zref-clever } { missing-hyperref }
158 { Missing~'hyperref'~package.~Setting~'hyperref=false'. }
159 \msg_new:nnn { zref-check } { check-document-only }
160 { Option~'check'~only~available~in~the~document. }
161 \msg_new:nnn { zref-clever } { missing-zref-check }
162 {
163   Option~'check'~requested~\msg_line_context:..
164   But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
165 }
166 \msg_new:nnn { zref-clever } { counters-not-nested }
167 { Counters~not~nested~for~labels~'#1'~and~'#2'~\msg_line_context:.. }
168 \msg_new:nnn { zref-clever } { missing-type }
169 { Reference~type~undefined~for~label~'#1'~\msg_line_context:.. }
170 \msg_new:nnn { zref-clever } { missing-name }
171 { Name~undefined~for~type~'#1'~\msg_line_context:.. }
172 \msg_new:nnn { zref-clever } { missing-string }
173 {
174   We~couldn't~find~a~value~for~reference~option~'#1'~\msg_line_context:..
175   But~we~should~have:~throw~a~rock~at~the~maintainer.
176 }
177 \msg_new:nnn { zref-clever } { single-element-range }
178 { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:.. }

```

4.2 Reference format

For a general discussion on the precedence rules for reference format options, see Section “Reference format” in the User manual. Internally, 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. The “fallback” settings are stored in

`\g_zrefclever_fallback_dict_prop.`

`\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 `\zcLanguageSetup`. But also for translations retrieval, in `_zrefclever_get_type-transl:nnnN` and `_zrefclever_get_default_transl:nnN`.

```
179 \tl_new:N \l_zrefclever_setup_type_tl
180 \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

181 \seq_const_from_clist:Nn
182   \c_zrefclever_ref_options_necessarily_not_type_specific_seq
183   {
184     tpairsep ,
185     tlistsep ,
186     tlastsep ,
187     notesep ,
188   }
189 \seq_const_from_clist:Nn
190   \c_zrefclever_ref_options_possibly_type_specific_seq
191   {
192     namesep ,
193     pairsep ,
194     listsep ,
195     lastsep ,
196     rangesep ,
197     refpre ,
198     refpos ,
199     refpre-in ,
200     refpos-in ,
201   }
```

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:`.

```
202 \seq_const_from_clist:Nn
203   \c_zrefclever_ref_options_necessarily_type_specific_seq
204   {
205     Name-sg ,
206     name-sg ,
207     Name-pl ,
208     name-pl ,
209     Name-sg-ab ,
210     name-sg-ab ,
211     Name-pl-ab ,
212     name-pl-ab ,
213   }
```

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

```

214 \seq_const_from_clist:Nn
215 \c__zrefclever_ref_options_font_seq
216 {
217     namefont ,
218     reffont ,
219     reffont-in ,
220 }
221 \seq_new:N \c__zrefclever_ref_options_typesetup_seq
222 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
223 \c__zrefclever_ref_options_possibly_type_specific_seq
224 \c__zrefclever_ref_options_necessarily_type_specific_seq
225 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
226 \c__zrefclever_ref_options_typesetup_seq
227 \c__zrefclever_ref_options_font_seq
228 \seq_new:N \c__zrefclever_ref_options_reference_seq
229 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
230 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
231 \c__zrefclever_ref_options_possibly_type_specific_seq
232 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
233 \c__zrefclever_ref_options_reference_seq
234 \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”.

```

235 \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 {\language}}

236 \NewDocumentCommand \zcDeclareLanguage { m }
237 {
238     \tl_if_empty:nF {#1}
239     {
240         \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
241         { \msg_warning:nnn { zref-clever } { language-declared } {#1} }
242         { \prop_gput:Nnn \g__zrefclever_languages_prop {#1} {#1} }
243     }
244 }
245 \@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}

246 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
247 {
248   \tl_if_empty:nF {#1}
249   {
250     \prop_if_in:NnTF \g__zrefclever_languages_prop {#2}
251     {
252       \exp_args:NnNx
253       \prop_gput:Nnn \g__zrefclever_languages_prop {#1}
254       { \prop_item:Nn \g__zrefclever_languages_prop {#2} }
255     }
256     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
257   }
258 }
259 \@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 `\zcLanguageSetup`, 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 `begindocument` 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 `begindocument`. This includes `translator`, `translations`, but also `babel`’s `.ldf` files, and `biblatex`’s `.ltx` 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 `\zcLanguageSetup` 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 `\zcLanguageSetup`, 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

`\g__zrefclever_loaded_dictionaries_seq` Used to keep track of whether a dictionary has already been loaded or not.

260 `\seq_new:N \g__zrefclever_loaded_dictionaries_seq`

(End definition for `\g__zrefclever_loaded_dictionaries_seq`.)

`\l__zrefclever_load_dict_verbose_bool` Controls whether `__zrefclever_provide_dictionary:n` fails silently or verbosely in case of unknown languages or dictionaries not found.

261 `\bool_new:N \l__zrefclever_load_dict_verbose_bool`

(End definition for `\l__zrefclever_load_dict_verbose_bool`.)

`__zrefclever_provide_dictionary:n` Load dictionary for known `⟨language⟩` if it is available and if it has not already been loaded.

```

\__zrefclever_provide_dictionary:n {⟨language⟩}
262 \cs_new_protected:Npn \__zrefclever_provide_dictionary:n #1
263 {
264   \group_begin:
265   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
266   \l__zrefclever_dict_language_tl
267   {
268     \seq_if_in:NVF
269     \g__zrefclever_loaded_dictionaries_seq
270     \l__zrefclever_dict_language_tl
271     {
272       \exp_args:Nx \file_get:nnNTF
273       { zref-clever- \l__zrefclever_dict_language_tl .dict }
274       { \ExplSyntaxOn }
275       \l_tmpa_tl
276       {
277         \prop_if_exist:cF
278         {
279           g__zrefclever_dict_
280           \l__zrefclever_dict_language_tl _prop
281         }
282         {

```

```

283         \prop_new:c
284         {
285             g__zrefclever_dict_
286             \l__zrefclever_dict_language_tl _prop
287         }
288     }
289     \tl_clear:N \l__zrefclever_setup_type_tl
290     \exp_args:NnV
291     \keys_set:nn { zref-clever / dictionary } \l_tmpa_tl
292     \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
293     \l__zrefclever_dict_language_tl
294     \msg_note:nnx { zref-clever } { dict-loaded }
295     { \l__zrefclever_dict_language_tl }
296 }
297 {
298     \bool_if:NT \l__zrefclever_load_dict_verbose_bool
299     {
300         \msg_warning:nnx { zref-clever } { dict-not-available }
301         { \l__zrefclever_dict_language_tl }
302     }

```

Even if we don't have the actual dictionary, we register it as “loaded”. At this point, it is a known language, properly declared. There is no point in trying to load it multiple times, because users cannot really provide the dictionary files (well, technically they could, but we are working so they don't need to, and have better ways to do what they want). And if the users had provided some translations themselves, by means of `\zcLanguageSetup`, everything would be in place, and they could use the `lang` option multiple times, and the `dict-not-available` warning would never go away.

```

303     \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
304     \l__zrefclever_dict_language_tl
305 }
306 }
307 }
308 {
309     \bool_if:NT \l__zrefclever_load_dict_verbose_bool
310     { \msg_warning:nnn { zref-clever } { unknown-language-load } {#1} }
311 }
312 \group_end:
313 }
314 \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>}

315 \cs_new_protected:Npn \__zrefclever_provide_dictionary_verbose:n #1
316 {
317     \group_begin:
318     \bool_set_true:N \l__zrefclever_load_dict_verbose_bool
319     \__zrefclever_provide_dictionary:n {#1}
320     \group_end:
321 }

```

```
322 \cs_generate_variant:Nn \__zrefclever_provide_dictionary_verbose:n { x }
```

(End definition for __zrefclever_provide_dictionary_verbose:n.)

__zrefclever_provide_dict_type_transl:nn A couple of auxiliary functions for the of zref-clever/dictionary keys set in
 __zrefclever_provide_dict_default_transl:nn __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 { \langle key \rangle } { \langle translation \rangle }
\__zrefclever_provide_dict_default_transl:nn { \langle key \rangle } { \langle translation \rangle }
```

```
323 \cs_new_protected:Npn \__zrefclever_provide_dict_type_transl:nn #1#2
324 {
325   \exp_args:Nnx \prop_gput_if_new:cnn
326     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
327     { type- \l__zrefclever_setup_type_tl - #1 } {#2}
328 }
329 \cs_new_protected:Npn \__zrefclever_provide_dict_default_transl:nn #1#2
330 {
331   \prop_gput_if_new:cnn
332     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
333     { default- #1 } {#2}
334 }
```

(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.

```
335 \keys_define:nn { zref-clever / dictionary }
336 {
337   type .code:n =
338   {
339     \tl_if_empty:NTF {#1}
340     { \tl_clear:N \l__zrefclever_setup_type_tl }
341     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
342   } ,
343 }
344 \seq_map_inline:Nn
345   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
346   {
347     \keys_define:nn { zref-clever / dictionary }
348     {
349       #1 .value_required:n = true ,
350       #1 .code:n =
351       {
352         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
353         { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
354         {
```

```

355         \msg_info:nnn { zref-clever }
356         { option-not-type-specific } {#1}
357     } ,
358 } ,
359 }
360 }
361 \seq_map_inline:Nn
362 \c__zrefclever_ref_options_possibly_type_specific_seq
363 {
364     \keys_define:nn { zref-clever / dictionary }
365     {
366         #1 .value_required:n = true ,
367         #1 .code:n =
368         {
369             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
370             { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
371             { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
372         } ,
373     }
374 }
375 \seq_map_inline:Nn
376 \c__zrefclever_ref_options_necessarily_type_specific_seq
377 {
378     \keys_define:nn { zref-clever / dictionary }
379     {
380         #1 .value_required:n = true ,
381         #1 .code:n =
382         {
383             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
384             {
385                 \msg_info:nnn { zref-clever }
386                 { option-only-type-specific } {#1}
387             }
388             { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
389         } ,
390     }
391 }

```

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.

```

392 \prop_new:N \g__zrefclever_fallback_dict_prop

```

```

393 \prop_gset_from_keyval:Nn \g__zrefclever_fallback_dict_prop
394 {
395     tpairsep = {,~} ,
396     tlistsep = {,~} ,
397     tlastsep = {,~} ,
398     notesep  = {~} ,
399     namesep  = {\nobreakspace} ,
400     pairsep  = {,~} ,
401     listsep  = {,~} ,
402     lastsep  = {,~} ,
403     rangesep = {\textendash} ,
404     refpre   = {} ,
405     refpos   = {} ,
406     refpre-in = {} ,
407     refpos-in = {} ,
408 }

```

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 {\langle language \rangle} {\langle type \rangle} {\langle key \rangle}
\langle tl variable \rangle {\langle false code \rangle}

409 \prg_new_protected_conditional:Npnn
410 \__zrefclever_get_type_transl:nnnN #1#2#3#4 { F }
411 {
412     \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
413     \l__zrefclever_dict_language_tl
414     {
415         \prop_get:cnNTF
416         { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
417         { type- #2 - #3 } #4
418         { \prg_return_true: }
419         { \prg_return_false: }
420     }
421     { \prg_return_false: }
422 }
423 \prg_generate_conditional_variant:Nnn
424 \__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 {\langle language \rangle} {\langle key \rangle}
\langle tl variable \rangle {\langle false code \rangle}

425 \prg_new_protected_conditional:Npnn
426 \__zrefclever_get_default_transl:nnN #1#2#3 { F }
427 {

```



```

428 \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
429 \l__zrefclever_dict_language_tl
430 {
431   \prop_get:cnNTF
432   { \g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
433   { default- #2 } #3
434   { \prg_return_true: }
435   { \prg_return_false: }
436 }
437 { \prg_return_false: }
438 }
439 \prg_generate_conditional_variant:Nnn
440 \__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>}

441 % {<key>}<tl var to set>
442 \prg_new_protected_conditional:Npnn
443 \__zrefclever_get_fallback_transl:nN #1#2 { F }
444 {
445   \prop_get:NnNTF \g__zrefclever_fallback_dict_prop
446   { #1 } #2
447   { \prg_return_true: }
448   { \prg_return_false: }
449 }

```

(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>}

450 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
451 {
452   \tl_if_empty:nTF {#3}
453   { \prop_remove:Nn #1 {#2} }
454   { \prop_put:Nnn #1 {#2} {#3} }
455 }

```

(End definition for __zrefclever_prop_put_non_empty:Nnn.)

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.

```

456 \tl_new:N \l__zrefclever_ref_property_tl
457 \keys_define:nn { zref-clever / reference }
458 {
459   ref .choice: ,
460   ref / zc@thecnt .code:n =
461     { \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt } } ,
462   ref / page .code:n =
463     { \tl_set:Nn \l__zrefclever_ref_property_tl { page } } ,
464   ref / title .code:n =
465     {
466       \AddToHook { begindocument }
467       {
468         \@ifpackageloaded { zref-titleref }
469         { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
470         {
471           \msg_warning:nn { zref-clever } { missing-zref-titleref }
472           \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
473         }
474       }
475     } ,
476   ref .initial:n = zc@thecnt ,
477   ref .default:n = zc@thecnt ,
478   page .meta:n = { ref = page },
479   page .value_forbidden:n = true ,
480 }
481 \AddToHook { begindocument }
482 {
483   \@ifpackageloaded { zref-titleref }
484   {
485     \keys_define:nn { zref-clever / reference }
486     {
487       ref / title .code:n =
488         { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
489     }
490   }
491   {
492     \keys_define:nn { zref-clever / reference }
493     {
494       ref / title .code:n =
495       {

```

```

496             \msg_warning:nn { zref-clever } { missing-zref-titleref }
497             \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
498         }
499     }
500 }
501 }

```

typeset option

```

502 \bool_new:N \l__zrefclever_typeset_ref_bool
503 \bool_new:N \l__zrefclever_typeset_name_bool
504 \keys_define:nn { zref-clever / reference }
505 {
506     typeset .choice: ,
507     typeset / both .code:n =
508     {
509         \bool_set_true:N \l__zrefclever_typeset_ref_bool
510         \bool_set_true:N \l__zrefclever_typeset_name_bool
511     } ,
512     typeset / ref .code:n =
513     {
514         \bool_set_true:N \l__zrefclever_typeset_ref_bool
515         \bool_set_false:N \l__zrefclever_typeset_name_bool
516     } ,
517     typeset / name .code:n =
518     {
519         \bool_set_false:N \l__zrefclever_typeset_ref_bool
520         \bool_set_true:N \l__zrefclever_typeset_name_bool
521     } ,
522     typeset .initial:n = both ,
523     typeset .value_required:n = true ,
524
525     noname .meta:n = { typeset = ref },
526     noname .value_forbidden:n = true ,
527 }

```

sort option

```

528 \bool_new:N \l__zrefclever_typeset_sort_bool
529 \keys_define:nn { zref-clever / reference }
530 {
531     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
532     sort .initial:n = true ,
533     sort .default:n = true ,
534     nosort .meta:n = { sort = false },
535     nosort .value_forbidden:n = true ,
536 }

```

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.

```

537 \seq_new:N \l__zrefclever_typesort_seq

```

```

538 \keys_define:nn { zref-clever / reference }
539 {
540   typesort .code:n =
541   {
542     \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
543     \seq_reverse:N \l__zrefclever_typesort_seq
544   } ,
545   typesort .initial:n =
546   { part , chapter , section , paragraph } ,
547   typesort .value_required:n = true ,
548   notypesort .code:n =
549   { \seq_clear:N \l__zrefclever_typesort_seq } ,
550   notypesort .value_forbidden:n = true ,
551 }

```

comp option

```

552 \bool_new:N \l__zrefclever_typeset_compress_bool
553 \keys_define:nn { zref-clever / reference }
554 {
555   comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
556   comp .initial:n = true ,
557   comp .default:n = true ,
558   nocomp .meta:n = { comp = false } ,
559   nocomp .value_forbidden:n = true ,
560 }

```

range option

```

561 \bool_new:N \l__zrefclever_typeset_range_bool
562 \keys_define:nn { zref-clever / reference }
563 {
564   range .bool_set:N = \l__zrefclever_typeset_range_bool ,
565   range .initial:n = false ,
566   range .default:n = true ,
567 }

```

cap and capfirst options

```

568 \bool_new:N \l__zrefclever_capitalize_bool
569 \bool_new:N \l__zrefclever_capitalize_first_bool
570 \keys_define:nn { zref-clever / reference }
571 {
572   cap .bool_set:N = \l__zrefclever_capitalize_bool ,
573   cap .initial:n = false ,
574   cap .default:n = true ,
575   nocap .meta:n = { cap = false } ,
576   nocap .value_forbidden:n = true ,
577
578   capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,
579   capfirst .initial:n = false ,
580   capfirst .default:n = true ,
581 }

```

abbrev and noabbrevfirst options

```

582 \bool_new:N \l__zrefclever_abbrev_bool

```

```

583 \bool_new:N \l__zrefclever_noabbrev_first_bool
584 \keys_define:nn { zref-clever / reference }
585 {
586   abbrev .bool_set:N = \l__zrefclever_abbrev_bool ,
587   abbrev .initial:n = false ,
588   abbrev .default:n = true ,
589   noabbrev .meta:n = { abbrev = false },
590   noabbrev .value_forbidden:n = true ,
591
592   noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
593   noabbrevfirst .initial:n = false ,
594   noabbrevfirst .default:n = true ,
595 }

```

S option

```

596 \keys_define:nn { zref-clever / reference }
597 {
598   S .meta:n =
599     { capfirst = true , noabbrevfirst = true },
600   S .value_forbidden:n = true ,
601 }

```

hyperref option

```

602 \bool_new:N \l__zrefclever_use_hyperref_bool
603 \bool_new:N \l__zrefclever_warn_hyperref_bool
604 \keys_define:nn { zref-clever / reference }
605 {
606   hyperref .choice: ,
607   hyperref / auto .code:n =
608     {
609       \bool_set_true:N \l__zrefclever_use_hyperref_bool
610       \bool_set_false:N \l__zrefclever_warn_hyperref_bool
611     } ,
612   hyperref / true .code:n =
613     {
614       \bool_set_true:N \l__zrefclever_use_hyperref_bool
615       \bool_set_true:N \l__zrefclever_warn_hyperref_bool
616     } ,
617   hyperref / false .code:n =
618     {
619       \bool_set_false:N \l__zrefclever_use_hyperref_bool
620       \bool_set_false:N \l__zrefclever_warn_hyperref_bool
621     } ,
622   hyperref .initial:n = auto ,
623   hyperref .default:n = auto
624 }
625 \AddToHook { begindocument }
626 {
627   \@ifpackageloaded { hyperref }
628   {
629     \bool_if:NT \l__zrefclever_use_hyperref_bool
630     { \RequirePackage { zref-hyperref } }
631   }
632   {

```

```

633     \bool_if:NT \l__zrefclever_warn_hyperref_bool
634     { \msg_warning:nn { zref-clever } { missing-hyperref } }
635     \bool_set_false:N \l__zrefclever_use_hyperref_bool
636   }
637   \keys_define:nn { zref-clever / reference }
638   {
639     hyperref .code:n =
640     { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }
641   }
642 }

```

nameinlink option

```

643 \str_new:N \l__zrefclever_nameinlink_str
644 \keys_define:nn { zref-clever / reference }
645 {
646   nameinlink .choice: ,
647   nameinlink / true .code:n =
648   { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
649   nameinlink / false .code:n =
650   { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
651   nameinlink / single .code:n =
652   { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
653   nameinlink / tsingle .code:n =
654   { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
655   nameinlink .initial:n = tsingle ,
656   nameinlink .default:n = true ,
657 }

```

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`.

```

658 \tl_new:N \l__zrefclever_ref_language_tl
659 \tl_new:N \l__zrefclever_main_language_tl
660 \tl_new:N \l__zrefclever_current_language_tl
661 \AddToHook { begindocument }
662 {
663   \@ifpackageloaded { babel }
664   {
665     \tl_set:Nn \l__zrefclever_current_language_tl { \language }
666     \tl_set:Nn \l__zrefclever_main_language_tl { \main@language }
667   }
668   {
669     \@ifpackageloaded { polyglossia }
670     {
671       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
672       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
673     }
674     {
675       \tl_set:Nn \l__zrefclever_current_language_tl { english }
676       \tl_set:Nn \l__zrefclever_main_language_tl { english }
677     }
678   }

```

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.

```

679   \tl_set:Nn \l__zrefclever_ref_language_tl
680   { \l__zrefclever_main_language_tl }
681 }
682 \keys_define:nn { zref-clever / reference }
683 {
684   lang .code:n =
685   {
686     \AddToHook { begindocument }
687     {
688       \str_case:nnF {#1}
689       {
690         { main }
691         {
692           \tl_set:Nn \l__zrefclever_ref_language_tl
693           { \l__zrefclever_main_language_tl }
694           \__zrefclever_provide_dictionary_verbosely:x
695           { \l__zrefclever_ref_language_tl }
696         }
697
698         { current }
699         {
700           \tl_set:Nn \l__zrefclever_ref_language_tl
701           { \l__zrefclever_current_language_tl }
702           \__zrefclever_provide_dictionary_verbosely:x

```

```

703         { \l__zrefclever_ref_language_tl }
704     }
705 }
706 {
707     \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
708     {
709         \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
710     }
711     {
712         \msg_warning:nnn { zref-clever }
713         { unknown-language-opt } {#1}
714         \tl_set:Nn \l__zrefclever_ref_language_tl
715         { \l__zrefclever_main_language_tl }
716     }
717     \__zrefclever_provide_dictionary_verbose:x
718     { \l__zrefclever_ref_language_tl }
719 }
720 }
721 } ,
722 lang .value_required:n = true ,
723 }
724 \AddToHook { begindocument / before }
725 {
726     \AddToHook { begindocument }
727     {

```

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.

```

728     \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }

```

Redefinition of the `lang` key option for the document body. Also, drop the verbose dictionary loading in the document body, as it can become intrusive depending on the use case, and does not provide much “juice” anyway: in `\zcref` missing names warnings will already ensue.

```

729     \keys_define:nn { zref-clever / reference }
730     {
731         lang .code:n =
732         {
733             \str_case:nnF {#1}
734             {
735                 { main }
736                 {
737                     \tl_set:Nn \l__zrefclever_ref_language_tl
738                     { \l__zrefclever_main_language_tl }
739                     \__zrefclever_provide_dictionary:x
740                     { \l__zrefclever_ref_language_tl }
741                 }
742
743                 { current }
744                 {
745                     \tl_set:Nn \l__zrefclever_ref_language_tl
746                     { \l__zrefclever_current_language_tl }
747                     \__zrefclever_provide_dictionary:x
748                     { \l__zrefclever_ref_language_tl }

```



```

749         }
750     }
751     {
752         \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
753         {
754             \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
755         }
756         {
757             \msg_warning:nnn { zref-clever }
758             { unknown-language-opt } {#1}
759             \tl_set:Nn \l__zrefclever_ref_language_tl
760             { \l__zrefclever_main_language_tl }
761         }
762         \__zrefclever_provide_dictionary:x
763         { \l__zrefclever_ref_language_tl }
764     }
765     } ,
766     lang .value_required:n = true ,
767 }
768 }
769 }

```

font option

`font` *can't be used as a package option*, since the options get expanded by L^AT_EX before being passed to the package (see <https://tex.stackexchange.com/a/489570>). It can't be set in `\zcref` and, for global settings, with `\zcsetup`.

```

770 \tl_new:N \l__zrefclever_ref_typeset_font_tl
771 \keys_define:nn { zref-clever / reference }
772 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

note option

```

773 \tl_new:N \l__zrefclever_zcref_note_tl
774 \keys_define:nn { zref-clever / reference }
775 {
776     note .tl_set:N = \l__zrefclever_zcref_note_tl ,
777     note .value_required:n = true ,
778 }

```

check option

Integration with `zref-check`.

```

779 \bool_new:N \l__zrefclever_zrefcheck_available_bool
780 \bool_new:N \l__zrefclever_zcref_with_check_bool
781 \keys_define:nn { zref-clever / reference }
782 {
783     check .code:n =
784     { \msg_warning:nnn { zref-clever } { check-document-only } } ,
785 }
786 \AddToHook { begindocument }
787 {
788     \@ifpackageloaded { zref-check }
789     {

```

```

790 \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
791 \keys_define:nn { zref-clever / reference }
792 {
793   check .code:n =
794   {
795     \bool_set_true:N \l__zrefclever_zcref_with_check_bool
796     \keys_set:nn { zref-check / zcheck } {#1}
797   }
798 }
799 }
800 {
801   \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
802   \keys_define:nn { zref-clever / reference }
803   {
804     check .code:n =
805     { \msg_warning:nn { zref-clever } { missing-zref-check } }
806   }
807 }
808 }

```

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`.

```

809 \prop_new:N \l__zrefclever_counter_type_prop
810 \keys_define:nn { zref-clever / label }
811 {
812   countertype .code:n =
813   {
814     \keyval_parse:nnn
815     {
816       \msg_warning:nnnn { zref-clever }
817       { key-requires-value } { countertype }
818     }
819     {
820       \__zrefclever_prop_put_non_empty:Nnn
821       \l__zrefclever_counter_type_prop
822     }
823     {#1}
824   } ,
825   countertype .value_required:n = true ,
826   countertype .initial:n =
827   {
828     subsection = section ,
829     subsubsection = section ,
830     subparagraph = paragraph ,
831     enumi = item ,
832     enumii = item ,
833     enumiii = item ,
834     enumiv = item ,
835   } ,

```

836 }

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.

```

837 \seq_new:N \l__zrefclever_counter_resetters_seq
838 \keys_define:nn { zref-clever / label }
839 {
840   counterresetters .code:n =
841   {
842     \clist_map_inline:nn {#1}
843     {
844       \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
845       {
846         \seq_put_right:Nn
847         \l__zrefclever_counter_resetters_seq {##1}
848       }
849     }
850   } ,
851   counterresetters .initial:n =
852   {
853     part ,
854     chapter ,
855     section ,
856     subsection ,
857     subsubsection ,
858     paragraph ,
859     subparagraph ,
860   },
861   counterresetters .value_required:n = true ,
862 }
```

counterresetby option

`\l__zrefclever_counter_resetby_prop` is used by `__zrefclever_counter_reset_by: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_reset_by:n` over the search through `\l__zrefclever_counter_resetters_seq`.

```

863 \prop_new:N \l__zrefclever_counter_resetby_prop
864 \keys_define:nn { zref-clever / label }
865 {
866   counterresetby .code:n =
867   {
868     \keyval_parse:nnn
```

```

869         {
870             \msg_warning:nnn { zref-clever }
871             { key-requires-value } { counterresetby }
872         }
873         {
874             \__zrefclever_prop_put_non_empty:Nnn
875             \l__zrefclever_counter_resetby_prop
876         }
877         {#1}
878     } ,
879     counterresetby .value_required:n = true ,
880     counterresetby .initial:n =
881     {

```

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.

```

882         enumii = enumi ,
883         enumiii = enumii ,
884         enumiv = enumiii ,
885     } ,
886 }

```

currentcounter option

`\l__zrefclever_current_counter_tl` is pretty much the starting point of all of the data specification for label setting done by `zref` with our setup for it. It exists because we must provide some “handle” to specify the current counter for packages/features that do not set `\@currentcounter` appropriately.

```

887 \tl_new:N \l__zrefclever_current_counter_tl
888 \keys_define:nn { zref-clever / label }
889 {
890     currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
891     currentcounter .value_required:n = true ,
892     currentcounter .initial:n = \@currentcounter ,
893 }

```

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.

```

894 \prop_new:N \l__zrefclever_ref_options_prop
895 \seq_map_inline:Nn

```

```

896 \c__zrefclever_ref_options_reference_seq
897 {
898   \keys_define:nn { zref-clever / reference }
899   {
900     #1 .default:V = \c_novalue_tl ,
901     #1 .code:n =
902     {
903       \tl_if_novalue:nTF {##1}
904       { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} }
905       { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
906     } ,
907   }
908 }

```

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.

```

909 \keys_define:nn { }
910 {
911   zref-clever / zcsetup .inherit:n =
912   {
913     zref-clever / label ,
914     zref-clever / reference ,
915   }
916 }

```

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

```

917 \ProcessKeysOptions { zref-clever / zcsetup }

```

5 Configuration

5.1 `\zcsetup`

`\zcsetup` Provide `\zcsetup`.

```

\zcsetup{<options>}

918 \NewDocumentCommand \zcsetup { m }
919 { \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 `\zcLanguageSetup` or by the package’s dictionaries. On the other hand, they have a lower precedence than non type-specific general options. The `<options>` should be given in the usual `key=val` format. The `<type>`

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

```
\zcRefTypeSetup      \zcRefTypeSetup {<type>} {<options>}
920 \NewDocumentCommand \zcRefTypeSetup { m m }
921 {
922   \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
923   { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
924   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
925   \keys_set:nn { zref-clever / typesetup } {#2}
926 }
```

(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>.

```
927 \seq_map_inline:Nn
928   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
929   {
930     \keys_define:nn { zref-clever / typesetup }
931     {
932       #1 .code:n =
933       {
934         \msg_warning:nnn { zref-clever }
935         { option-not-type-specific } {#1}
936       } ,
937     }
938   }
939 \seq_map_inline:Nn
940   \c__zrefclever_ref_options_typesetup_seq
941   {
942     \keys_define:nn { zref-clever / typesetup }
943     {
944       #1 .default:V = \c_novaluel_tl ,
945       #1 .code:n =
946       {
947         \tl_if_novalue:nTF {##1}
948         {
949           \prop_remove:cn
950           {
951             l__zrefclever_type_
```

```

952         \l__zrefclever_setup_type_tl _options_prop
953     }
954     {#1}
955 }
956 {
957     \prop_put:cnn
958     {
959         l__zrefclever_type_
960         \l__zrefclever_setup_type_tl _options_prop
961     }
962     {#1} {##1}
963 }
964 } ,
965 }
966 }

```

5.3 \zcLanguageSetup

\zcLanguageSetup is the main user interface for “language-specific” reference formatting, be it “type-specific” or not. The difference between the two cases is captured by the `type` key, which works as a sort of a “switch”. Inside the `<options>` argument of \zcLanguageSetup, 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. \zcLanguageSetup is preamble only.

```

\zcLanguageSetup      \zcLanguageSetup{<language>}{<options>}
967 \NewDocumentCommand \zcLanguageSetup { m m }
968 {
969     \group_begin:
970     \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
971     \l__zrefclever_dict_language_tl
972     {
973         \tl_clear:N \l__zrefclever_setup_type_tl
974         \keys_set:nn { zref-clever / langsetup } {#2}
975     }
976     { \msg_warning:nnn { zref-clever } { unknown-language-transl } {#1} }
977     \group_end:
978 }
979 \@onlypreamble \zcLanguageSetup

```

(End definition for \zcLanguageSetup.)

```

\__zrefclever_declare_type_transl:nnnn
\__zrefclever_declare_default_transl:nnn

```

A couple of auxiliary functions for the of `zref-clever/translation` keys set in \zcLanguageSetup. 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>}

```

```

980 \cs_new_protected:Npn \__zrefclever_declare_type_transl:nnnn #1#2#3#4
981 {
982   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
983   { type- #2 - #3 } {#4}
984 }
985 \cs_generate_variant:Nn \__zrefclever_declare_type_transl:nnnn { VVnn }
986 \cs_new_protected:Npn \__zrefclever_declare_default_transl:nnn #1#2#3
987 {
988   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
989   { default- #2 } {#3}
990 }
991 \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/langsetup, which is used to set language-specific translations in \zcLanguageSetup.

```

992 \keys_define:nn { zref-clever / langsetup }
993 {
994   type .code:n =
995   {
996     \tl_if_empty:NTF {#1}
997     { \tl_clear:N \l__zrefclever_setup_type_tl }
998     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
999   } ,
1000 }
1001 \seq_map_inline:Nn
1002 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
1003 {
1004   \keys_define:nn { zref-clever / langsetup }
1005   {
1006     #1 .value_required:n = true ,
1007     #1 .code:n =
1008     {
1009       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1010       {
1011         \__zrefclever_declare_default_transl:Vnn
1012         \l__zrefclever_dict_language_tl
1013         {#1} {##1}
1014       }
1015       {
1016         \msg_warning:nnn { zref-clever }
1017         { option-not-type-specific } {#1}
1018       }
1019     } ,
1020   }
1021 }
1022 \seq_map_inline:Nn
1023 \c__zrefclever_ref_options_possibly_type_specific_seq
1024 {
1025   \keys_define:nn { zref-clever / langsetup }
1026   {
1027     #1 .value_required:n = true ,
1028     #1 .code:n =

```



```

1029         {
1030             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1031             {
1032                 \__zrefclever_declare_default_transl:Vnn
1033                 \l__zrefclever_dict_language_tl
1034                 {#1} {##1}
1035             }
1036             {
1037                 \__zrefclever_declare_type_transl:Vnn
1038                 \l__zrefclever_dict_language_tl
1039                 \l__zrefclever_setup_type_tl
1040                 {#1} {##1}
1041             }
1042         } ,
1043     }
1044 }
1045 \seq_map_inline:Nn
1046 \c__zrefclever_ref_options_necessarily_type_specific_seq
1047 {
1048     \keys_define:nn { zref-clever / langsetup }
1049     {
1050         #1 .value_required:n = true ,
1051         #1 .code:n =
1052         {
1053             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1054             {
1055                 \msg_warning:nnn { zref-clever }
1056                 { option-only-type-specific } {#1}
1057             }
1058             {
1059                 \__zrefclever_declare_type_transl:Vnn
1060                 \l__zrefclever_dict_language_tl
1061                 \l__zrefclever_setup_type_tl
1062                 {#1} {##1}
1063             }
1064         } ,
1065     }
1066 }

```

6 User interface

6.1 \zcref

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

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

```

1067 \NewDocumentCommand \zcref { s O { } m }
1068 { \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>}}

```

```

1069 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
1070 {
1071   \group_begin:

```

Set options.

```

1072   \keys_set:nn { zref-clever / reference } {#3}

```

Store arguments values.

```

1073   \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
1074   \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.

```

1075   \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }

```

Integration with `zref-check`.

```

1076   \bool_lazy_and:nnT
1077     { \l__zrefclever_zrefcheck_available_bool }
1078     { \l__zrefclever_zcref_with_check_bool }
1079     { \zrefcheck_zcref_beg_label: }

```

Sort the labels.

```

1080   \bool_lazy_or:nnT
1081     { \l__zrefclever_typeset_sort_bool }
1082     { \l__zrefclever_typeset_range_bool }
1083     { \__zrefclever_sort_labels: }

```

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

```

1084   \group_begin:
1085   \l__zrefclever_ref_typeset_font_tl
1086   \__zrefclever_typeset_refs:
1087   \group_end:

```

Typeset note.

```

1088   \tl_if_empty:NF \l__zrefclever_zcref_note_tl
1089   {
1090     \__zrefclever_get_ref_string:nN { notesep } \l_tmpa_tl
1091     \l_tmpa_tl
1092     \l__zrefclever_zcref_note_tl
1093   }

```

Integration with `zref-check`.

```

1094   \bool_lazy_and:nnT
1095     { \l__zrefclever_zrefcheck_available_bool }
1096     { \l__zrefclever_zcref_with_check_bool }
1097     {
1098       \zrefcheck_zcref_end_label_maybe:
1099       \zrefcheck_zcref_run_checks_on_labels:n
1100         { \l__zrefclever_zcref_labels_seq }
1101     }
1102   \group_end:
1103 }

```

(End definition for `_zrefclever_zcref:nnnn`.)

`\l_zrefclever_zcref_labels_seq`
`\l_zrefclever_link_star_bool`

```
1104 \seq_new:N \l__zrefclever_zcref_labels_seq
1105 \bool_new:N \l__zrefclever_link_star_bool
```

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

6.2 `\zcpageref`

`\zcpageref` A `\pageref` equivalent of `\zcref`.

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

```
1106 \NewDocumentCommand \zcpageref { s O { } m }
1107 {
1108   \IfBooleanTF {#1}
1109     { \zcref*[#2, ref = page] {#3} }
1110     { \zcref [ #2, ref = page] {#3} }
1111 }
```

(End definition for `\zcpageref`.)

7 Sorting

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

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

`\l_zrefclever_label_type_a_tl`
`\l_zrefclever_label_type_b_tl`
`\l_zrefclever_label_enclcnt_a_tl`
`\l_zrefclever_label_enclcnt_b_tl`
`\l_zrefclever_label_enclval_a_tl`
`\l_zrefclever_label_enclval_b_tl`

```
1112 \tl_new:N \l__zrefclever_label_type_a_tl
1113 \tl_new:N \l__zrefclever_label_type_b_tl
1114 \tl_new:N \l__zrefclever_label_enclcnt_a_tl
1115 \tl_new:N \l__zrefclever_label_enclcnt_b_tl
1116 \tl_new:N \l__zrefclever_label_enclval_a_tl
1117 \tl_new:N \l__zrefclever_label_enclval_b_tl
```

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

`\l_zrefclever_sort_decided_bool`

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

```
1118 \bool_new:N \l__zrefclever_sort_decided_bool
```

(End definition for `\l__zrefclever_sort_decided_bool`.)

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

```
1119 \int_new:N \l__zrefclever_sort_prior_a_int
1120 \int_new:N \l__zrefclever_sort_prior_b_int
```

(End definition for `\l__zrefclever_sort_prior_a_int` and `\l__zrefclever_sort_prior_b_int`.)

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

```
1121 \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.

```
1122 \cs_new_protected:Npn \__zrefclever_sort_labels:
1123 {
```

Store label types sequence.

```
1124   \seq_clear:N \l__zrefclever_label_types_seq
1125   \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
1126   {
1127     \seq_map_function:NN \l__zrefclever_zcref_labels_seq
1128     \__zrefclever_label_type_put_new_right:n
1129   }
```

Sort.

```
1130   \seq_sort:Nn \l__zrefclever_zcref_labels_seq
1131   {
1132     \zref@ifrefundefined {##1}
1133     {
1134       \zref@ifrefundefined {##2}
1135       {
1136         % Neither label is defined.
1137         \sort_return_same:
1138       }
1139       {
1140         % The second label is defined, but the first isn't, leave the
1141         % undefined first (to be more visible).
1142         \sort_return_same:
1143       }
1144     }
1145     {
1146       \zref@ifrefundefined {##2}
1147       {
1148         % The first label is defined, but the second isn't, bring the
1149         % second forward.
1150         \sort_return_swapped:
```

```

1151     }
1152     {
1153         % The interesting case: both labels are defined. References
1154         % to the "default" property or to the "page" are quite
1155         % different with regard to sorting, so we branch them here to
1156         % specialized functions.
1157         \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1158             { \__zrefclever_sort_page:nn {##1} {##2} }
1159             { \__zrefclever_sort_default:nn {##1} {##2} }
1160     }
1161 }
1162 }
1163 }

```

(End definition for __zrefclever_sort_labels:.)

__zrefclever_label_type_put_new_right:n

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

```

\__zrefclever_label_type_put_new_right:n {<label>}

1164 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
1165 {
1166     \tl_set:Nx \l__zrefclever_label_type_a_tl
1167     { \zref@extractdefault {#1} {zc@type} { \c_empty_tl } }
1168     \seq_if_in:NVF \l__zrefclever_label_types_seq
1169     \l__zrefclever_label_type_a_tl
1170     {
1171         \seq_put_right:NV \l__zrefclever_label_types_seq
1172         \l__zrefclever_label_type_a_tl
1173     }
1174 }

```

(End definition for __zrefclever_label_type_put_new_right:n.)

__zrefclever_sort_default:nn

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

```

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

1175 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
1176 {
1177     \tl_set:Nx \l__zrefclever_label_type_a_tl
1178     { \zref@extractdefault {#1} {zc@type} { \c_empty_tl } }
1179     \tl_set:Nx \l__zrefclever_label_type_b_tl
1180     { \zref@extractdefault {#2} {zc@type} { \c_empty_tl } }

```

```

1181
1182 \bool_if:nTF
1183 {
1184   % The second label has a type, but the first doesn't, leave the
1185   % undefined first (to be more visible).
1186   \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1187   ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1188 }
1189 { \sort_return_same: }
1190 {
1191   \bool_if:nTF
1192   {
1193     % The first label has a type, but the second doesn't, bring the
1194     % second forward.
1195     ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1196     \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1197   }
1198   { \sort_return_swapped: }
1199   {
1200     \bool_if:nTF
1201     {
1202       % The interesting case: both labels have a type...
1203       ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1204       ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1205     }
1206     {
1207       \tl_if_eq:nNTF
1208       \l__zrefclever_label_type_a_tl
1209       \l__zrefclever_label_type_b_tl
1210       % ...and it's the same type.
1211       { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
1212       % ...and they are different types.
1213       { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
1214     }
1215     {
1216       % Neither label has a type. We can't do much of meaningful
1217       % here, but if it's the same counter, compare it.
1218       \exp_args:Nxx \tl_if_eq:nnTF
1219       { \zref@extractdefault {#1} { zc@counter } { } }
1220       { \zref@extractdefault {#2} { zc@counter } { } }
1221       {
1222         \int_compare:nNnTF
1223         { \zref@extractdefault {#1} { zc@cntval } { -1 } }
1224         >
1225         { \zref@extractdefault {#2} { zc@cntval } { -1 } }
1226         { \sort_return_swapped: }
1227         { \sort_return_same: }
1228       }
1229       { \sort_return_same: }
1230     }
1231   }
1232 }
1233 }

```

(End definition for `__zrefclever_sort_default:nn`.)

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

```

1234 \cs_generate_variant:Nn \tl_reverse_items:n { V }

\_zrefclever_sort_default_same_type:nn      \__zrefclever_sort_default_same_type:nn {\label a} {\label b}
1235 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
1236 {
1237   \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1238     { \zref@extractdefault {#1} { zc@enclcnt } { \c_empty_tl } }
1239   \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1240     { \tl_reverse_items:V \l__zrefclever_label_enclcnt_a_tl }
1241   \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1242     { \zref@extractdefault {#2} { zc@enclcnt } { \c_empty_tl } }
1243   \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1244     { \tl_reverse_items:V \l__zrefclever_label_enclcnt_b_tl }
1245   \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1246     { \zref@extractdefault {#1} { zc@enclval } { \c_empty_tl } }
1247   \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1248     { \tl_reverse_items:V \l__zrefclever_label_enclval_a_tl }
1249   \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1250     { \zref@extractdefault {#2} { zc@enclval } { \c_empty_tl } }
1251   \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1252     { \tl_reverse_items:V \l__zrefclever_label_enclval_b_tl }
1253
1254   \bool_set_false:N \l__zrefclever_sort_decided_bool
1255   \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1256     {
1257       \bool_if:nTF
1258         {
1259           % Both are empty: neither label has any (further) "enclosing
1260           % counters" (left).
1261           \tl_if_empty_p:V \l__zrefclever_label_enclcnt_a_tl &&
1262           \tl_if_empty_p:V \l__zrefclever_label_enclcnt_b_tl
1263         }
1264         {
1265           \exp_args:Nxx \tl_if_eq:nnTF
1266             { \zref@extractdefault {#1} { zc@counter } { } }
1267             { \zref@extractdefault {#2} { zc@counter } { } }
1268             {
1269               \bool_set_true:N \l__zrefclever_sort_decided_bool
1270               \int_compare:nNnTF
1271                 { \zref@extractdefault {#1} { zc@cntval } { -1 } }
1272                 >
1273                 { \zref@extractdefault {#2} { zc@cntval } { -1 } }
1274                 { \sort_return_swapped: }
1275                 { \sort_return_same: }
1276             }
1277         }
1278       \msg_warning:nnnn { zref-clever }
1279         { counters-not-nested } {#1} {#2}
1280       \bool_set_true:N \l__zrefclever_sort_decided_bool
1281       \sort_return_same:
1282     }

```

```

1283 }
1284 {
1285     \bool_if:nTF
1286     {
1287         % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
1288         \tl_if_empty_p:V \l__zrefclever_label_enclcnt_a_tl
1289     }
1290     {
1291         \int_zero:N \l_tmpb_int
1292         \tl_map_inline:Nn \l__zrefclever_label_enclcnt_b_tl
1293         {
1294             \int_incr:N \l_tmpb_int
1295             \exp_args:Nnx \tl_if_eq:nnT {##1}
1296             { \zref@extractdefault {#1} { zc@counter } { } }
1297             {
1298                 \tl_map_break:n
1299                 {
1300                     \int_compare:nNnTF
1301                     { \zref@extractdefault {#1} { zc@cntval } { } }
1302                     >
1303                     {
1304                         \tl_item:Nn \l__zrefclever_label_enclval_b_tl
1305                         { \l_tmpb_int }
1306                     }
1307                     { \sort_return_swapped: }
1308                     { \sort_return_same: }
1309                     \bool_set_true:N \l__zrefclever_sort_decided_bool
1310                 }
1311             }
1312         }
1313         \bool_if:NF \l__zrefclever_sort_decided_bool
1314         {
1315             \msg_warning:nnnn { zref-clever }
1316             { counters-not-nested } {#1} {#2}
1317             \bool_set_true:N \l__zrefclever_sort_decided_bool
1318             \sort_return_same:
1319         }
1320     }
1321 }
1322 {
1323     \bool_if:nTF
1324     {
1325         % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
1326         \tl_if_empty_p:V \l__zrefclever_label_enclcnt_b_tl
1327     }
1328     {
1329         \int_zero:N \l_tmpa_int
1330         \tl_map_inline:Nn \l__zrefclever_label_enclcnt_a_tl
1331         {
1332             \int_incr:N \l_tmpa_int
1333             \exp_args:Nnx \tl_if_eq:nnT {##1}
1334             { \zref@extractdefault {#2} { zc@counter } { } }
1335             {
1336                 \tl_map_break:n
1337                 {

```



```

1337         \int_compare:nNnTF
1338         {
1339             \tl_item:Nn
1340             \l__zrefclever_label_enclval_a_tl
1341             { \l_tmpa_int }
1342         }
1343         <
1344         {
1345             \zref@extractdefault {#2}
1346             { zc@cntval } { }
1347         }
1348         { \sort_return_same: }
1349         { \sort_return_swapped: }
1350     \bool_set_true:N
1351     \l__zrefclever_sort_decided_bool
1352 }
1353 }
1354 }
1355 \bool_if:NF \l__zrefclever_sort_decided_bool
1356 {
1357     \msg_warning:nnnn { zref-clever }
1358     { counters-not-nested } {#1} {#2}
1359     \bool_set_true:N \l__zrefclever_sort_decided_bool
1360     \sort_return_same:
1361 }
1362 }
1363 {
1364     % Neither is empty: we can (possibly) compare the values
1365     % of the current enclosing counter in the loop, if they
1366     % are equal, we are still in the loop, if they are not, a
1367     % sorting decision can be made directly.
1368     \exp_args:Nxx \tl_if_eq:nnTF
1369     { \tl_head:N \l__zrefclever_label_enclcnt_a_tl }
1370     { \tl_head:N \l__zrefclever_label_enclcnt_b_tl }
1371     {
1372         \int_compare:nNnTF
1373         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1374         =
1375         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1376         {
1377             \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1378             { \tl_tail:N \l__zrefclever_label_enclcnt_a_tl }
1379             \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1380             { \tl_tail:N \l__zrefclever_label_enclcnt_b_tl }
1381             \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1382             { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1383             \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1384             { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1385         }
1386         {
1387             \bool_set_true:N \l__zrefclever_sort_decided_bool
1388             \int_compare:nNnTF
1389             { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1390             >

```

```

1391         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1392         { \sort_return_swapped: }
1393         { \sort_return_same: }
1394     }
1395 }
1396 {
1397     \msg_warning:nnnn { zref-clever }
1398     { counters-not-nested } {#1} {#2}
1399     \bool_set_true:N \l__zrefclever_sort_decided_bool
1400     \sort_return_same:
1401 }
1402 }
1403 }
1404 }
1405 }
1406 }

```

(End definition for `__zrefclever_sort_default_same_type:nn`.)

```

_zrefclever_sort_default_different_types:nn    \__zrefclever_sort_default_different_types:nn {<label a>} {<label b>}
1407 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
1408 {

```

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

```

1409     \int_zero:N \l__zrefclever_sort_prior_a_int
1410     \int_zero:N \l__zrefclever_sort_prior_b_int
1411     \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
1412     {
1413         \tl_if_eq:nnTF {##2} {{othertypes}}
1414         {
1415             \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
1416             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1417             \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
1418             { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1419         }
1420         {
1421             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
1422             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1423             {
1424                 \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
1425                 { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1426             }
1427         }
1428     }

```

Then do the actual sorting.

```

1429     \bool_if:nTF
1430     {
1431         \int_compare_p:nNn
1432         { \l__zrefclever_sort_prior_a_int } <
1433         { \l__zrefclever_sort_prior_b_int }
1434     }

```

```

1435 { \sort_return_same: }
1436 {
1437   \bool_if:nTF
1438   {
1439     \int_compare_p:nNn
1440     { \l__zrefclever_sort_prior_a_int } >
1441     { \l__zrefclever_sort_prior_b_int }
1442   }
1443   { \sort_return_swapped: }
1444   {
1445     % Sort priorities are equal: the type that occurs first in
1446     % ‘labels’, as given by the user, is kept (or brought) forward.
1447     \seq_map_inline:Nn \l__zrefclever_label_types_seq
1448     {
1449       \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1450       { \seq_map_break:n { \sort_return_same: } }
1451       {
1452         \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
1453         { \seq_map_break:n { \sort_return_swapped: } }
1454       }
1455     }
1456   }
1457 }
1458 }

```

(End definition for `__zrefclever_sort_default_different_types:nn`.)

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

```

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

1459 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
1460 {
1461   \int_compare:nNnTF
1462   { \zref@extractdefault {#1} { abspage } {-1} }
1463   >
1464   { \zref@extractdefault {#2} { abspage } {-1} }
1465   { \sort_return_swapped: }
1466   { \sort_return_same: }
1467 }

```

(End definition for `__zrefclever_sort_page:nn`.)

8 Typesetting

“Typesetting” the reference, which here includes the parsing of the labels and eventual compression of labels in sequence into ranges, is definitely the “crux” of `zref-clever`. This because we process the label set as a stack, in a single pass, and hence “parsing”, “compressing”, and “typesetting” must be decided upon at the same time, making it difficult to slice the job into more specific and self-contained tasks. So, do bear this in mind before

you curse me for the length of some of the functions below, or before a more orthodox “docstripper” complains about me not sticking to code commenting conventions to keep the code more readable in the .dtx file.

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

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

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

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

One further thing to discuss here – to keep this “on record” – is inhibition of compression for individual labels. It is not difficult to handle it at the infrastructure side, what

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

Variables

Auxiliary variables for `__zrefclever_typeset_refs`: main stack control.

```
\l_zrefclever_typeset_labels_seq
\l_zrefclever_typeset_last_bool
\l_zrefclever_last_of_type_bool
1468 \seq_new:N \l__zrefclever_typeset_labels_seq
1469 \bool_new:N \l_zrefclever_typeset_last_bool
1470 \bool_new:N \l__zrefclever_last_of_type_bool
```

(End definition for `\l_zrefclever_typeset_labels_seq`, `\l_zrefclever_typeset_last_bool`, and `\l__zrefclever_last_of_type_bool`.)

Auxiliary variables for `__zrefclever_typeset_refs`: main counters.

```
\l_zrefclever_type_count_int
\l_zrefclever_label_count_int
1471 \int_new:N \l__zrefclever_type_count_int
1472 \int_new:N \l_zrefclever_label_count_int
```

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

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

```
\l__zrefclever_label_a_tl
\l__zrefclever_label_b_tl
\l_zrefclever_typeset_queue_prev_tl
\l_zrefclever_typeset_queue_curr_tl
\l_zrefclever_type_first_label_tl
\l_zrefclever_type_first_label_type_tl
1473 \tl_new:N \l__zrefclever_label_a_tl
1474 \tl_new:N \l__zrefclever_label_b_tl
1475 \tl_new:N \l_zrefclever_typeset_queue_prev_tl
1476 \tl_new:N \l_zrefclever_typeset_queue_curr_tl
1477 \tl_new:N \l_zrefclever_type_first_label_tl
1478 \tl_new:N \l_zrefclever_type_first_label_type_tl
```

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

Auxiliary variables for `__zrefclever_typeset_refs`: type name handling.

```
\l_zrefclever_type_name_tl
\l_zrefclever_name_in_link_bool
\l_zrefclever_name_format_tl
\l_zrefclever_name_format_fallback_tl
1479 \tl_new:N \l__zrefclever_type_name_tl
1480 \bool_new:N \l_zrefclever_name_in_link_bool
1481 \tl_new:N \l_zrefclever_name_format_tl
1482 \tl_new:N \l_zrefclever_name_format_fallback_tl
```

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

Auxiliary variables for `__zrefclever_typeset_refs`: range handling.

```
\l_zrefclever_range_count_int
\l_zrefclever_range_same_count_int
\l_zrefclever_range_beg_label_tl
\l_zrefclever_next_maybe_range_bool
\l_zrefclever_next_is_same_bool
1483 \int_new:N \l__zrefclever_range_count_int
1484 \int_new:N \l_zrefclever_range_same_count_int
1485 \tl_new:N \l_zrefclever_range_beg_label_tl
1486 \bool_new:N \l_zrefclever_next_maybe_range_bool
1487 \bool_new:N \l_zrefclever_next_is_same_bool
```

(End definition for \l__zrefclever_range_count_int and others.)

Auxiliary variables for __zrefclever_typeset_refs: separators, refpre/pos and font options.

```

\l__zrefclever_tpairsep_tl
\l__zrefclever_tlistsep_tl
\l__zrefclever_tlastsep_tl
\l__zrefclever_namesep_tl
\l__zrefclever_pairsep_tl
\l__zrefclever_listsep_tl
\l__zrefclever_lastsep_tl
\l__zrefclever_rangesep_tl
\l__zrefclever_refpre_out_tl
\l__zrefclever_refpos_out_tl
\l__zrefclever_refpre_in_tl
\l__zrefclever_refpos_in_tl
\l__zrefclever_namefont_tl
\l__zrefclever_reffont_out_tl
\l__zrefclever_reffont_in_tl

```

(End definition for \l__zrefclever_tpairsep_tl and others.)

Main functions

__zrefclever_typeset_refs: Main typesetting function for \zceref.

```

1503 \cs_new_protected:Npn \__zrefclever_typeset_refs:
1504 {
1505   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
1506   \l__zrefclever_zceref_labels_seq
1507   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
1508   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1509   \tl_clear:N \l__zrefclever_type_first_label_tl
1510   \tl_clear:N \l__zrefclever_type_first_label_type_tl
1511   \tl_clear:N \l__zrefclever_range_beg_label_tl
1512   \int_zero:N \l__zrefclever_label_count_int
1513   \int_zero:N \l__zrefclever_type_count_int
1514   \int_zero:N \l__zrefclever_range_count_int
1515   \int_zero:N \l__zrefclever_range_same_count_int
1516
1517   % Get type block options (not type-specific).
1518   \__zrefclever_get_ref_string:nN { tpairsep }
1519   \l__zrefclever_tpairsep_tl
1520   \__zrefclever_get_ref_string:nN { tlistsep }
1521   \l__zrefclever_tlistsep_tl
1522   \__zrefclever_get_ref_string:nN { tlastsep }
1523   \l__zrefclever_tlastsep_tl
1524
1525   % Process label stack.
1526   \bool_set_false:N \l__zrefclever_typeset_last_bool
1527   \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1528   {
1529     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
1530     \l__zrefclever_label_a_tl
1531     \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq

```

```

1532     {
1533         \tl_clear:N \l__zrefclever_label_b_tl
1534         \bool_set_true:N \l__zrefclever_typeset_last_bool
1535     }
1536     {
1537         \seq_get_left:NN \l__zrefclever_typeset_labels_seq
1538         \l__zrefclever_label_b_tl
1539     }
1540
1541 \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1542 {
1543     \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1544     \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1545 }
1546 {
1547     \tl_set:Nx \l__zrefclever_label_type_a_tl
1548     {
1549         \zref@extractdefault
1550         { \l__zrefclever_label_a_tl } { zc@type } { \c_empty_tl }
1551     }
1552     \tl_set:Nx \l__zrefclever_label_type_b_tl
1553     {
1554         \zref@extractdefault
1555         { \l__zrefclever_label_b_tl } { zc@type } { \c_empty_tl }
1556     }
1557 }
1558
1559 % First, we establish whether the "current label" (i.e. 'a') is the
1560 % last one of its type. This can happen because the "next label"
1561 % (i.e. 'b') is of a different type (or different definition status),
1562 % or because we are at the end of the list.
1563 \bool_if:NnTF \l__zrefclever_typeset_last_bool
1564 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1565 {
1566     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1567     {
1568         \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1569         { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1570         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1571     }
1572     {
1573         \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1574         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1575         {
1576             % Neither is undefined, we must check the types.
1577             \bool_if:nTF
1578             {
1579                 % Both empty: same "type".
1580                 \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1581                 \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1582             }
1583             { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1584             {
1585                 \bool_if:nTF

```

```

1586         {
1587             % Neither empty: compare types.
1588             ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl
1589             &&
1590             ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1591         }
1592         {
1593             \tl_if_eq:NNTF
1594             \l__zrefclever_label_type_a_tl
1595             \l__zrefclever_label_type_b_tl
1596             {
1597                 \bool_set_false:N
1598                 \l__zrefclever_last_of_type_bool
1599             }
1600             {
1601                 \bool_set_true:N
1602                 \l__zrefclever_last_of_type_bool
1603             }
1604         }
1605         % One empty, the other not: different "types".
1606         {
1607             \bool_set_true:N
1608             \l__zrefclever_last_of_type_bool
1609         }
1610     }
1611 }
1612 }
1613 }
1614
1615 % Handle warnings in case of reference or type undefined.
1616 \zref@refused { \l__zrefclever_label_a_tl }
1617 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1618 {}
1619 {
1620     \tl_if_empty:NT \l__zrefclever_label_type_a_tl
1621     {
1622         \msg_warning:nxx { zref-clever } { missing-type }
1623         { \l__zrefclever_label_a_tl }
1624     }
1625 }
1626
1627 % Get type-specific separators, refpre/pos and font options, once per
1628 % type.
1629 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
1630 {
1631     \__zrefclever_get_ref_string:nN { namesep      }
1632     \l__zrefclever_namesep_tl
1633     \__zrefclever_get_ref_string:nN { rangesep     }
1634     \l__zrefclever_rangesep_tl
1635     \__zrefclever_get_ref_string:nN { pairsep      }
1636     \l__zrefclever_pairsep_tl
1637     \__zrefclever_get_ref_string:nN { listsep      }
1638     \l__zrefclever_listsep_tl
1639     \__zrefclever_get_ref_string:nN { lastsep      }

```



```

1640         \l__zrefclever_lastsep_tl
1641         \__zrefclever_get_ref_string:nN { refpre      }
1642         \l__zrefclever_refpre_out_tl
1643         \__zrefclever_get_ref_string:nN { refpos      }
1644         \l__zrefclever_refpos_out_tl
1645         \__zrefclever_get_ref_string:nN { refpre-in   }
1646         \l__zrefclever_refpre_in_tl
1647         \__zrefclever_get_ref_string:nN { refpos-in   }
1648         \l__zrefclever_refpos_in_tl
1649         \__zrefclever_get_ref_font:nN   { namefont    }
1650         \l__zrefclever_namefont_tl
1651         \__zrefclever_get_ref_font:nN   { reffont     }
1652         \l__zrefclever_reffont_out_tl
1653         \__zrefclever_get_ref_font:nN   { reffont-in  }
1654         \l__zrefclever_reffont_in_tl
1655     }
1656
1657     % Here we send this to a couple of auxiliary functions.
1658     \bool_if:NTF \l__zrefclever_last_of_type_bool
1659     { % There exists no next label of the same type as the current.
1660       { \__zrefclever_typeset_refs_last_of_type: }
1661       % There exists a next label of the same type as the current.
1662       { \__zrefclever_typeset_refs_not_last_of_type: }
1663     }
1664 }

```

(End definition for __zrefclever_typeset_refs:.)

This is actually the one meaningful “big branching” we can do while processing the label stack: i) the “current” label is the last of its type block; or ii) the “current” label is *not* the last of its type block. Indeed, as mentioned above, quite a number of things can only be decided when the type block ends, and we only know this when we look at the “next” label and find something of a different “type” (loose here, maybe different definition status, maybe end of stack). So, though this is not very strict, __zrefclever_typeset_refs_last_of_type: is more of a “wrapping up” function, and it is indeed the one which does the actual typesetting, while __zrefclever_typeset_refs_not_last_of_type: is more of an “accumulation” function.

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

```

1665 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
1666 {
1667     % Process the current label to the current queue.
1668     \int_case:nnF { \l__zrefclever_label_count_int }
1669     {
1670         % It is the last label of its type, but also the first one, and that's
1671         % what matters here: just store it.
1672         { 0 }
1673         {
1674             \tl_set:NV \l__zrefclever_type_first_label_tl
1675             \l__zrefclever_label_a_tl
1676             \tl_set:NV \l__zrefclever_type_first_label_type_tl
1677             \l__zrefclever_label_type_a_tl
1678         }
1679
1680         % The last is the second: we have a pair (if not repeated).

```

```

1681 { 1 }
1682 {
1683   \int_compare:nNnF { \l__zrefclever_range_same_count_int } = { 1 }
1684   {
1685     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1686     {
1687       \exp_not:V \l__zrefclever_pairsep_tl
1688       \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1689     }
1690   }
1691 }
1692 }
1693 % Last is third or more of its type: without repetition, we'd have the
1694 % last element on a list, but control for possible repetition.
1695 {
1696   \int_case:nnF { \l__zrefclever_range_count_int }
1697   {
1698     % There was no range going on.
1699     { 0 }
1700     {
1701       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1702       {
1703         \exp_not:V \l__zrefclever_lastsep_tl
1704         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1705       }
1706     }
1707     % Last in the range is also the second in it.
1708     { 1 }
1709     {
1710       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1711       {
1712         % We know 'range_beg_label' is not empty, since this is the
1713         % second element in the range, but the third or more in the
1714         % type list.
1715         \exp_not:V \l__zrefclever_listsep_tl
1716         \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1717         \int_compare:nNnF
1718         { \l__zrefclever_range_same_count_int } = { 1 }
1719         {
1720           \exp_not:V \l__zrefclever_lastsep_tl
1721           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1722         }
1723       }
1724     }
1725   }
1726   % Last in the range is third or more in it.
1727   {
1728     \int_case:nnF
1729     {
1730       \l__zrefclever_range_count_int -
1731       \l__zrefclever_range_same_count_int
1732     }
1733     {
1734       % Repetition, not a range.

```

```

1735 { 0 }
1736 {
1737   % If 'range_beg_label' is empty, it means it was also the
1738   % first of the type, and hence was already handled.
1739   \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1740   {
1741     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1742     {
1743       \exp_not:V \l__zrefclever_lastsep_tl
1744       \__zrefclever_get_ref:V
1745       \l__zrefclever_range_beg_label_tl
1746     }
1747   }
1748 }
1749 % A 'range', but with no skipped value, treat as list.
1750 { 1 }
1751 {
1752   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1753   {
1754     % Ditto.
1755     \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1756     {
1757       \exp_not:V \l__zrefclever_listsep_tl
1758       \__zrefclever_get_ref:V
1759       \l__zrefclever_range_beg_label_tl
1760     }
1761     \exp_not:V \l__zrefclever_lastsep_tl
1762     \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1763   }
1764 }
1765 }
1766 {
1767   % An actual range.
1768   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1769   {
1770     % Ditto.
1771     \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1772     {
1773       \exp_not:V \l__zrefclever_lastsep_tl
1774       \__zrefclever_get_ref:V
1775       \l__zrefclever_range_beg_label_tl
1776     }
1777     \exp_not:V \l__zrefclever_rangesep_tl
1778     \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1779   }
1780 }
1781 }
1782 }
1783
1784 % Handle "range" option. The idea is simple: if the queue is not empty,
1785 % we replace it with the end of the range (or pair). We can still
1786 % retrieve the end of the range from 'label_a' since we know to be
1787 % processing the last label of its type at this point.
1788 \bool_if:NT \l__zrefclever_typeset_range_bool

```

```

1789 {
1790   \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
1791   {
1792     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1793     { }
1794     {
1795       \msg_warning:nxx { zref-clever } { single-element-range }
1796       { \l__zrefclever_type_first_label_type_tl }
1797     }
1798   }
1799   {
1800     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1801     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1802     { }
1803     {
1804       \__zrefclever_labels_in_sequence:nn
1805       { \l__zrefclever_type_first_label_tl }
1806       { \l__zrefclever_label_a_tl }
1807     }
1808     \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1809     {
1810       \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1811       { \exp_not:V \l__zrefclever_pairsep_tl }
1812       { \exp_not:V \l__zrefclever_rangesep_tl }
1813       \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1814     }
1815   }
1816 }
1817
1818 % Now that the type block is finished, we can add the name and the first
1819 % ref to the queue. Also, if "typeset" option is not "both", handle it
1820 % here as well.
1821 \__zrefclever_type_name_setup:
1822 \bool_if:NTF
1823 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
1824 {
1825   \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1826   { \__zrefclever_get_ref_first: }
1827 }
1828 {
1829   \bool_if:NTF
1830   { \l__zrefclever_typeset_ref_bool }
1831   {
1832     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1833     { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
1834   }
1835   {
1836     \bool_if:NTF
1837     { \l__zrefclever_typeset_name_bool }
1838     {
1839       \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1840       {
1841         \bool_if:NTF \l__zrefclever_name_in_link_bool
1842         {

```

```

1843 \exp_not:N \group_begin:
1844 \exp_not:V \l__zrefclever_namefont_tl
1845 % It's two '@s', but escaped for DocStrip.
1846 \exp_not:N \hyper@@link
1847 {
1848   \zref@ifrefcontainsprop
1849   { \l__zrefclever_type_first_label_tl }
1850   { urluse }
1851   {
1852     \zref@extractdefault
1853     { \l__zrefclever_type_first_label_tl }
1854     { urluse } {}
1855   }
1856   {
1857     \zref@extractdefault
1858     { \l__zrefclever_type_first_label_tl }
1859     { url } {}
1860   }
1861 }
1862 {
1863   \zref@extractdefault
1864   { \l__zrefclever_type_first_label_tl }
1865   { anchor } {}
1866 }
1867 { \exp_not:V \l__zrefclever_type_name_tl }
1868 \exp_not:N \group_end:
1869 }
1870 {
1871   \exp_not:N \group_begin:
1872   \exp_not:V \l__zrefclever_namefont_tl
1873   \exp_not:V \l__zrefclever_type_name_tl
1874   \exp_not:N \group_end:
1875 }
1876 }
1877 }
1878 {
1879   % Logically, this case would correspond to "typeset=none", but
1880   % it should not occur, given that the options are set up to
1881   % typeset either "ref" or "name". Still, leave here a
1882   % sensible fallback, equal to the behavior of "both".
1883   \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1884     { \__zrefclever_get_ref_first: }
1885 }
1886 }
1887 }
1888
1889 % Typeset the previous type, if there is one.
1890 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
1891 {
1892   \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
1893   { \l__zrefclever_tlistsep_tl }
1894   \l__zrefclever_typeset_queue_prev_tl
1895 }
1896

```

```

1897 % Wrap up loop, or prepare for next iteration.
1898 \bool_if:NTF \l__zrefclever_typeset_last_bool
1899 {
1900   % We are finishing, typeset the current queue.
1901   \int_case:nnF { \l__zrefclever_type_count_int }
1902   {
1903     % Single type.
1904     { 0 }
1905     { \l__zrefclever_typeset_queue_curr_tl }
1906     % Pair of types.
1907     { 1 }
1908     {
1909       \l__zrefclever_tpairsep_tl
1910       \l__zrefclever_typeset_queue_curr_tl
1911     }
1912   }
1913   {
1914     % Last in list of types.
1915     \l__zrefclever_tlastsep_tl
1916     \l__zrefclever_typeset_queue_curr_tl
1917   }
1918 }
1919 {
1920   % There are further labels, set variables for next iteration.
1921   \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
1922     \l__zrefclever_typeset_queue_curr_tl
1923   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1924   \tl_clear:N \l__zrefclever_type_first_label_tl
1925   \tl_clear:N \l__zrefclever_type_first_label_type_tl
1926   \tl_clear:N \l__zrefclever_range_beg_label_tl
1927   \int_zero:N \l__zrefclever_label_count_int
1928   \int_incr:N \l__zrefclever_type_count_int
1929   \int_zero:N \l__zrefclever_range_count_int
1930   \int_zero:N \l__zrefclever_range_same_count_int
1931 }
1932 }

```

(End definition for _zrefclever_typeset_refs_last_of_type:.)

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

```

1933 \cs_new_protected:Npn \_zrefclever_typeset_refs_not_last_of_type:
1934 {
1935   % Signal if next label may form a range with the current one (only
1936   % considered if compression is enabled in the first place).
1937   \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1938   \bool_set_false:N \l__zrefclever_next_is_same_bool
1939   \bool_if:NT \l__zrefclever_typeset_compress_bool
1940   {
1941     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1942     { }
1943     {
1944       \_zrefclever_labels_in_sequence:nn
1945       { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
1946     }
1947   }

```

```

1947     }
1948
1949 % Process the current label to the current queue.
1950 \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
1951 {
1952     % Current label is the first of its type (also not the last, but it
1953     % doesn't matter here): just store the label.
1954     \tl_set:NV \l__zrefclever_type_first_label_tl
1955         \l__zrefclever_label_a_tl
1956     \tl_set:NV \l__zrefclever_type_first_label_type_tl
1957         \l__zrefclever_label_type_a_tl
1958
1959     % If the next label may be part of a range, we set 'range_beg_label'
1960     % to "empty" (we deal with it as the "first", and must do it there, to
1961     % handle hyperlinking), but also step the range counters.
1962     \bool_if:NT \l__zrefclever_next_maybe_range_bool
1963     {
1964         \tl_clear:N \l__zrefclever_range_beg_label_tl
1965         \int_incr:N \l__zrefclever_range_count_int
1966         \bool_if:NT \l__zrefclever_next_is_same_bool
1967             { \int_incr:N \l__zrefclever_range_same_count_int }
1968     }
1969 }
1970 {
1971     % Current label is neither the first (nor the last) of its type.
1972     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1973     {
1974         % Starting, or continuing a range.
1975         \int_compare:nNnTF
1976             { \l__zrefclever_range_count_int } = { 0 }
1977         {
1978             % There was no range going, we are starting one.
1979             \tl_set:NV \l__zrefclever_range_beg_label_tl
1980                 \l__zrefclever_label_a_tl
1981             \int_incr:N \l__zrefclever_range_count_int
1982             \bool_if:NT \l__zrefclever_next_is_same_bool
1983                 { \int_incr:N \l__zrefclever_range_same_count_int }
1984         }
1985         {
1986             % Second or more in the range, but not the last.
1987             \int_incr:N \l__zrefclever_range_count_int
1988             \bool_if:NT \l__zrefclever_next_is_same_bool
1989                 { \int_incr:N \l__zrefclever_range_same_count_int }
1990         }
1991     }
1992 }
1993 % Next element is not in sequence: there was no range, or we are
1994 % closing one.
1995 \int_case:nnF { \l__zrefclever_range_count_int }
1996 {
1997     % There was no range going on.
1998     { 0 }
1999     {
2000         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl

```

```

2001         {
2002             \exp_not:V \l__zrefclever_listsep_tl
2003             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2004         }
2005     }
2006     % Last is second in the range: if 'range_same_count' is also
2007     % '1', it's a repetition (drop it), otherwise, it's a "pair
2008     % within a list", treat as list.
2009     { 1 }
2010     {
2011         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2012         {
2013             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2014             {
2015                 \exp_not:V \l__zrefclever_listsep_tl
2016                 \__zrefclever_get_ref:V
2017                 \l__zrefclever_range_beg_label_tl
2018             }
2019             \int_compare:nNnF
2020             { \l__zrefclever_range_same_count_int } = { 1 }
2021             {
2022                 \exp_not:V \l__zrefclever_listsep_tl
2023                 \__zrefclever_get_ref:V
2024                 \l__zrefclever_label_a_tl
2025             }
2026         }
2027     }
2028 }
2029 {
2030     % Last is third or more in the range: if 'range_count' and
2031     % 'range_same_count' are the same, its a repetition (drop it),
2032     % if they differ by '1', its a list, if they differ by more,
2033     % it is a real range.
2034     \int_case:nnF
2035     {
2036         \l__zrefclever_range_count_int -
2037         \l__zrefclever_range_same_count_int
2038     }
2039     {
2040         { 0 }
2041         {
2042             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2043             {
2044                 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2045                 {
2046                     \exp_not:V \l__zrefclever_listsep_tl
2047                     \__zrefclever_get_ref:V
2048                     \l__zrefclever_range_beg_label_tl
2049                 }
2050             }
2051         }
2052         { 1 }
2053         {
2054             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl

```



```

2055         {
2056             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2057             {
2058                 \exp_not:V \l__zrefclever_listsep_tl
2059                 \__zrefclever_get_ref:V
2060                 \l__zrefclever_range_beg_label_tl
2061             }
2062             \exp_not:V \l__zrefclever_listsep_tl
2063             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2064         }
2065     }
2066 }
2067 {
2068     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2069     {
2070         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2071         {
2072             \exp_not:V \l__zrefclever_listsep_tl
2073             \__zrefclever_get_ref:V
2074             \l__zrefclever_range_beg_label_tl
2075         }
2076         \exp_not:V \l__zrefclever_rangeseq_tl
2077         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2078     }
2079 }
2080 }
2081 % Reset counters.
2082 \int_zero:N \l__zrefclever_range_count_int
2083 \int_zero:N \l__zrefclever_range_same_count_int
2084 }
2085 }
2086 % Step label counter for next iteration.
2087 \int_incr:N \l__zrefclever_label_count_int
2088 }

```

(End definition for `__zrefclever_typeset_refs_not_last_of_type:`.)

Aux functions

`__zrefclever_get_ref:n` and `__zrefclever_get_ref_first:` are the two functions which actually build the reference blocks for typesetting. `__zrefclever_get_ref:n` handles all references but the first of its type, and `__zrefclever_get_ref_first:` deals with the first reference of a type. Saying they do “typesetting” is imprecise though, they actually prepare material to be accumulated in `\l__zrefclever_typeset_queue_curr_tl` inside `__zrefclever_typeset_refs_last_of_type:` and `__zrefclever_typeset_refs_not_last_of_type:`. And this difference results quite crucial for the \TeX nic requirements of these functions. This because, as we are processing the label stack and accumulating content in the queue, we are using a number of variables which are transient to the current label, the label properties among them, but not only. Hence, these variables *must* be expanded to their current values to be stored in the queue. Indeed, `__zrefclever_get_ref:n` and `__zrefclever_get_ref_first:` get called, as they must, in the context of x type expansions. But we don’t want to expand the values of the variables themselves, so we need to get current values, but stop expansion after

that. In particular, reference options given by the user should reach the stream for its final typesetting (when the queue itself gets typeset) *unmodified* (“no manipulation”, to use the `n` signature jargon). We also need to prevent premature expansion of material that can’t be expanded at this point (e.g. grouping, `\zref@default` or `\hyper@@link`). In a nutshell, the job of these two functions is putting the pieces in place, but with proper expansion control.

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

```
2089 \cs_new_protected:Npn \__zrefclever_ref_default:
2090 { \zref@default }
2091 \cs_new_protected:Npn \__zrefclever_name_default:
2092 { \zref@default }
```

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

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

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

2093 \cs_new:Npn \__zrefclever_get_ref:n #1
2094 {
2095   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
2096   {
2097     \bool_if:nTF
2098     {
2099       \l__zrefclever_use_hyperref_bool &&
2100       ! \l__zrefclever_link_star_bool
2101     }
2102     {
2103       \exp_not:N \group_begin:
2104       \exp_not:V \l__zrefclever_reffont_out_tl
2105       \exp_not:V \l__zrefclever_refpre_out_tl
2106       \exp_not:N \group_begin:
2107       \exp_not:V \l__zrefclever_reffont_in_tl
2108       % It’s two ‘@s’, but escaped for DocStrip.
2109       \exp_not:N \hyper@@link
2110       {
2111         \zref@ifrefcontainsprop {#1} { urluse }
2112         { \zref@extractdefault {#1} { urluse } { } }
2113         { \zref@extractdefault {#1} { url } { } }
2114       }
2115       { \zref@extractdefault {#1} { anchor } { } }
2116       {
2117         \exp_not:V \l__zrefclever_refpre_in_tl
2118         \zref@extractdefault {#1}
2119         { \l__zrefclever_ref_property_tl } { }
2120         \exp_not:V \l__zrefclever_refpos_in_tl
2121       }
2122       \exp_not:N \group_end:
```

```

2123         \exp_not:V \l__zrefclever_refpos_out_tl
2124         \exp_not:N \group_end:
2125     }
2126     {
2127         \exp_not:N \group_begin:
2128         \exp_not:V \l__zrefclever_reffont_out_tl
2129         \exp_not:V \l__zrefclever_refpre_out_tl
2130         \exp_not:N \group_begin:
2131         \exp_not:V \l__zrefclever_reffont_in_tl
2132         \exp_not:V \l__zrefclever_refpre_in_tl
2133         \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } { }
2134         \exp_not:V \l__zrefclever_refpos_in_tl
2135         \exp_not:N \group_end:
2136         \exp_not:V \l__zrefclever_refpos_out_tl
2137         \exp_not:N \group_end:
2138     }
2139 }
2140 { \__zrefclever_ref_default: }
2141 }
2142 \cs_generate_variant:Nn \__zrefclever_get_ref:n { V }

```

(End definition for __zrefclever_get_ref:n.)

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

```

2143 \cs_new:Npn \__zrefclever_get_ref_first:
2144 {
2145     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2146     { \__zrefclever_ref_default: }
2147     {
2148         \bool_if:NTF \l__zrefclever_name_in_link_bool
2149         {
2150             \zref@ifrefcontainsprop
2151             { \l__zrefclever_type_first_label_tl }
2152             { \l__zrefclever_ref_property_tl }
2153             {
2154                 % It's two '@s', but escaped for DocStrip.
2155                 \exp_not:N \hyper@@link
2156                 {
2157                     \zref@ifrefcontainsprop
2158                     { \l__zrefclever_type_first_label_tl } { urluse }
2159                     {
2160                         \zref@extractdefault
2161                         { \l__zrefclever_type_first_label_tl }
2162                         { urluse } { }
2163                     }
2164                 }
2165                 \zref@extractdefault

```

```

2166         { \l__zrefclever_type_first_label_tl }
2167         { url } { }
2168     }
2169 }
2170 {
2171     \zref@extractdefault
2172     { \l__zrefclever_type_first_label_tl }
2173     { anchor } { }
2174 }
2175 {
2176     \exp_not:N \group_begin:
2177     \exp_not:V \l__zrefclever_namefont_tl
2178     \exp_not:V \l__zrefclever_type_name_tl
2179     \exp_not:N \group_end:
2180     \exp_not:V \l__zrefclever_namesep_tl
2181     \exp_not:N \group_begin:
2182     \exp_not:V \l__zrefclever_reffont_out_tl
2183     \exp_not:V \l__zrefclever_refpre_out_tl
2184     \exp_not:N \group_begin:
2185     \exp_not:V \l__zrefclever_reffont_in_tl
2186     \exp_not:V \l__zrefclever_refpre_in_tl
2187     \zref@extractdefault
2188     { \l__zrefclever_type_first_label_tl }
2189     { \l__zrefclever_ref_property_tl } { }
2190     \exp_not:V \l__zrefclever_refpos_in_tl
2191     \exp_not:N \group_end:
2192     % hyperlink makes it's own group, we'd like to close the
2193     % 'refpre-out' group after 'refpos-out', but... we close
2194     % it here, and give the trailing 'refpos-out' its own
2195     % group. This will result that formatting given to
2196     % 'refpre-out' will not reach 'refpos-out', but I see no
2197     % alternative, and this has to be handled specially.
2198     \exp_not:N \group_end:
2199 }
2200 \exp_not:N \group_begin:
2201 % Ditto: special treatment.
2202 \exp_not:V \l__zrefclever_reffont_out_tl
2203 \exp_not:V \l__zrefclever_refpos_out_tl
2204 \exp_not:N \group_end:
2205 }
2206 {
2207     \exp_not:N \group_begin:
2208     \exp_not:V \l__zrefclever_namefont_tl
2209     \exp_not:V \l__zrefclever_type_name_tl
2210     \exp_not:N \group_end:
2211     \exp_not:V \l__zrefclever_namesep_tl
2212     \__zrefclever_ref_default:
2213 }
2214 }
2215 {
2216     \tl_if_empty:NTF \l__zrefclever_type_name_tl
2217     {
2218         \__zrefclever_name_default:
2219         \exp_not:V \l__zrefclever_namesep_tl

```

```

2220 }
2221 {
2222   \exp_not:N \group_begin:
2223   \exp_not:V \l__zrefclever_namefont_tl
2224   \exp_not:V \l__zrefclever_type_name_tl
2225   \exp_not:N \group_end:
2226   \exp_not:V \l__zrefclever_namesep_tl
2227 }
2228 \zref@ifrefcontainsprop
2229 { \l__zrefclever_type_first_label_tl }
2230 { \l__zrefclever_ref_property_tl }
2231 {
2232   \bool_if:nTF
2233   {
2234     \l__zrefclever_use_hyperref_bool &&
2235     ! \l__zrefclever_link_star_bool
2236   }
2237   {
2238     \exp_not:N \group_begin:
2239     \exp_not:V \l__zrefclever_reffont_out_tl
2240     \exp_not:V \l__zrefclever_refpre_out_tl
2241     \exp_not:N \group_begin:
2242     \exp_not:V \l__zrefclever_reffont_in_tl
2243     % It's two '@s', but escaped for DocStrip.
2244     \exp_not:N \hyper@@link
2245     {
2246       \zref@ifrefcontainsprop
2247       { \l__zrefclever_type_first_label_tl } { urluse }
2248       {
2249         \zref@extractdefault
2250         { \l__zrefclever_type_first_label_tl }
2251         { urluse } { }
2252       }
2253       {
2254         \zref@extractdefault
2255         { \l__zrefclever_type_first_label_tl }
2256         { url } { }
2257       }
2258     }
2259     {
2260       \zref@extractdefault
2261       { \l__zrefclever_type_first_label_tl }
2262       { anchor } { }
2263     }
2264     {
2265       \exp_not:V \l__zrefclever_refpre_in_tl
2266       \zref@extractdefault
2267       { \l__zrefclever_type_first_label_tl }
2268       { \l__zrefclever_ref_property_tl } { }
2269       \exp_not:V \l__zrefclever_refpos_in_tl
2270     }
2271     \exp_not:N \group_end:
2272     \exp_not:V \l__zrefclever_refpos_out_tl
2273     \exp_not:N \group_end:

```

```

2274     }
2275     {
2276         \exp_not:N \group_begin:
2277         \exp_not:V \l__zrefclever_reffont_out_tl
2278         \exp_not:V \l__zrefclever_refpre_out_tl
2279         \exp_not:N \group_begin:
2280         \exp_not:V \l__zrefclever_reffont_in_tl
2281         \exp_not:V \l__zrefclever_refpre_in_tl
2282         \zref@extractdefault
2283         { \l__zrefclever_type_first_label_tl }
2284         { \l__zrefclever_ref_property_tl } { }
2285         \exp_not:V \l__zrefclever_refpos_in_tl
2286         \exp_not:N \group_end:
2287         \exp_not:V \l__zrefclever_refpos_out_tl
2288         \exp_not:N \group_end:
2289     }
2290 }
2291 { \__zrefclever_ref_default: }
2292 }
2293 }
2294 }

```

(End definition for __zrefclever_get_ref_first:.)

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

```

2295 \cs_new_protected:Npn \__zrefclever_type_name_setup:
2296 {
2297     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2298     { \tl_clear:N \l__zrefclever_type_name_tl }
2299     {
2300         \tl_if_empty:NTF \l__zrefclever_type_first_label_type_tl
2301         { \tl_clear:N \l__zrefclever_type_name_tl }
2302         {
2303             % Determine whether we should use capitalization, abbreviation,
2304             % and plural.
2305             \bool_lazy_or:nnTF
2306             { \l__zrefclever_capitalize_bool }
2307             {
2308                 \l__zrefclever_capitalize_first_bool &&
2309                 \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2310             }
2311             { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
2312             { \tl_set:Nn \l__zrefclever_name_format_tl {name} }

```

```

2313 % If the queue is empty, we have a singular, otherwise, plural.
2314 \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2315 { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
2316 { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
2317 \bool_lazy_and:nnTF
2318 { \l__zrefclever_abbrev_bool }
2319 {
2320 ! \int_compare_p:nNn
2321 { \l__zrefclever_type_count_int } = { 0 } ||
2322 ! \l__zrefclever_noabbrev_first_bool
2323 }
2324 {
2325 \tl_set:NV \l__zrefclever_name_format_fallback_tl
2326 \l__zrefclever_name_format_tl
2327 \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
2328 }
2329 { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
2330
2331 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
2332 {
2333 \prop_get:cVNF
2334 {
2335 \l__zrefclever_type_
2336 \l__zrefclever_type_first_label_type_tl _options_prop
2337 }
2338 \l__zrefclever_name_format_tl
2339 \l__zrefclever_type_name_tl
2340 {
2341 \__zrefclever_get_type_transl:xxxNF
2342 { \l__zrefclever_ref_language_tl }
2343 { \l__zrefclever_type_first_label_type_tl }
2344 { \l__zrefclever_name_format_tl }
2345 \l__zrefclever_type_name_tl
2346 {
2347 \tl_clear:N \l__zrefclever_type_name_tl
2348 \msg_warning:nxx { zref-clever } { missing-name }
2349 { \l__zrefclever_type_first_label_type_tl }
2350 }
2351 }
2352 }
2353 {
2354 \prop_get:cVNF
2355 {
2356 \l__zrefclever_type_
2357 \l__zrefclever_type_first_label_type_tl _options_prop
2358 }
2359 \l__zrefclever_name_format_tl
2360 \l__zrefclever_type_name_tl
2361 {
2362 \prop_get:cVNF
2363 {
2364 \l__zrefclever_type_
2365 \l__zrefclever_type_first_label_type_tl _options_prop
2366 }

```

```

2367         \l__zrefclever_name_format_fallback_tl
2368         \l__zrefclever_type_name_tl
2369         {
2370             \__zrefclever_get_type_transl:xxxNF
2371             { \l__zrefclever_ref_language_tl }
2372             { \l__zrefclever_type_first_label_type_tl }
2373             { \l__zrefclever_name_format_tl }
2374             \l__zrefclever_type_name_tl
2375             {
2376                 \__zrefclever_get_type_transl:xxxNF
2377                 { \l__zrefclever_ref_language_tl }
2378                 { \l__zrefclever_type_first_label_type_tl }
2379                 { \l__zrefclever_name_format_fallback_tl }
2380                 \l__zrefclever_type_name_tl
2381                 {
2382                     \tl_clear:N \l__zrefclever_type_name_tl
2383                     \msg_warning:nnx { zref-clever }
2384                     { missing-name }
2385                     { \l__zrefclever_type_first_label_type_tl }
2386                 }
2387             }
2388         }
2389     }
2390 }
2391
2392 }
2393
2394 % Signal whether the type name is to be included in the hyperlink or not.
2395 \bool_lazy_any:nTF
2396 {
2397     { ! \l__zrefclever_use_hyperref_bool }
2398     { \l__zrefclever_link_star_bool }
2399     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
2400     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
2401 }
2402 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2403 {
2404     \bool_lazy_any:nTF
2405     {
2406         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
2407         {
2408             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
2409             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2410         }
2411         {
2412             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
2413             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2414             \l__zrefclever_typeset_last_bool &&
2415             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2416         }
2417     }
2418     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2419     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2420 }

```


2421 }

(End definition for _zrefclever_type_name_setup:.)

_zrefclever_labels_in_sequence:nn

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

_zrefclever_labels_in_sequence:nn {\langle label\ a \rangle} {\langle label\ b \rangle}

```

2422 \cs_new_protected:Npn \_zrefclever_labels_in_sequence:nn #1#2
2423 {
2424   \tl_if_eq:NnTF \l\_zrefclever_ref_property_tl { page }
2425   {
2426     \exp_args:Nxx \tl_if_eq:nnT
2427     { \zref@extractdefault {#1} { zc@pgfmt } { } }
2428     { \zref@extractdefault {#2} { zc@pgfmt } { } }
2429     {
2430       \int_compare:nNnTF
2431       { \zref@extractdefault {#1} { zc@pgval } { -2 } + 1 }
2432       =
2433       { \zref@extractdefault {#2} { zc@pgval } { -1 } }
2434       { \bool_set_true:N \l\_zrefclever_next_maybe_range_bool }
2435       {
2436         \int_compare:nNnT
2437         { \zref@extractdefault {#1} { zc@pgval } { -1 } }
2438         =
2439         { \zref@extractdefault {#2} { zc@pgval } { -1 } }
2440         {
2441           \bool_set_true:N \l\_zrefclever_next_maybe_range_bool
2442           \bool_set_true:N \l\_zrefclever_next_is_same_bool
2443         }
2444       }
2445     }
2446   }
2447   {
2448     \exp_args:Nxx \tl_if_eq:nnT
2449     { \zref@extractdefault {#1} { zc@counter } { } }
2450     { \zref@extractdefault {#2} { zc@counter } { } }
2451     {
2452       \exp_args:Nxx \tl_if_eq:nnT
2453       { \zref@extractdefault {#1} { zc@enclval } { } }
2454       { \zref@extractdefault {#2} { zc@enclval } { } }
2455       {
2456         \int_compare:nNnTF
2457         { \zref@extractdefault {#1} { zc@cntval } { -2 } + 1 }
2458         =
2459         { \zref@extractdefault {#2} { zc@cntval } { -1 } }
2460         { \bool_set_true:N \l\_zrefclever_next_maybe_range_bool }
2461         {
2462           \int_compare:nNnT

```

```

2463         { \zref@extractdefault {#1} { zc@cntval } { -1 } }
2464         =
2465         { \zref@extractdefault {#2} { zc@cntval } { -1 } }
2466         {
2467             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2468             \bool_set_true:N \l__zrefclever_next_is_same_bool
2469         }
2470     }
2471 }
2472 }
2473 }
2474 }

```

(End definition for `__zrefclever_labels_in_sequence:nn`.)

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

```

\__zrefclever_get_ref_string:nN      \__zrefclever_get_ref_string:nN {<option>} {<tl variable>}
2475 \cs_new_protected:Npn \__zrefclever_get_ref_string:nN #1#2
2476 {
2477     % First attempt: general options.
2478     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2479     {
2480         % If not found, try type specific options.
2481         \bool_lazy_all:nTF
2482         {
2483             { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2484             {
2485                 \prop_if_exist_p:c
2486                 {
2487                     l__zrefclever_type_
2488                     \l__zrefclever_label_type_a_tl _options_prop
2489                 }
2490             }
2491             {
2492                 \prop_if_in_p:cn
2493                 {
2494                     l__zrefclever_type_
2495                     \l__zrefclever_label_type_a_tl _options_prop
2496                 }
2497                 {#1}
2498             }
2499         }

```

```

2500     {
2501       \prop_get:cnN
2502       {
2503         l__zrefclever_type_
2504         \l__zrefclever_label_type_a_tl _options_prop
2505       }
2506       {#1} #2
2507     }
2508     {
2509       % If not found, try type specific translations.
2510       \__zrefclever_get_type_transl:xnNF
2511       { \l__zrefclever_ref_language_tl }
2512       { \l__zrefclever_label_type_a_tl }
2513       {#1} #2
2514       {
2515         % If not found, try default translations.
2516         \__zrefclever_get_default_transl:xnNF
2517         { \l__zrefclever_ref_language_tl }
2518         {#1} #2
2519         {
2520           % If not found, try fallback.
2521           \__zrefclever_get_fallback_transl:nNF {#1} #2
2522           {
2523             \tl_clear:N #2
2524             \msg_warning:nnn { zref-clever }
2525             { missing-string } {#1}
2526           }
2527         }
2528       }
2529     }
2530   }
2531 }

```

(End definition for __zrefclever_get_ref_string:nN.)

```

\__zrefclever_get_ref_font:nN      \__zrefclever_get_ref_font:nN {<option>} {<tl variable>}
2532 \cs_new_protected:Npn \__zrefclever_get_ref_font:nN #1#2
2533 {
2534   % First attempt: general options.
2535   \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2536   {
2537     % If not found, try type specific options.
2538     \bool_lazy_and:nnTF
2539     { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2540     {
2541       \prop_if_exist_p:c
2542       {
2543         l__zrefclever_type_
2544         \l__zrefclever_label_type_a_tl _options_prop
2545       }
2546     }
2547     {
2548       \prop_get:cnNF
2549       {

```

```

2550         l__zrefclever_type_
2551         \l__zrefclever_label_type_a_tl _options_prop
2552     }
2553     {#1} #2
2554     { \tl_clear:N #2 }
2555 }
2556 { \tl_clear:N #2 }
2557 }
2558 }

```

(End definition for `_zrefclever_get_ref_font:nN`.)

9 Compatibility

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`

One relevant case of different reference types sharing 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.

The standard `\appendix` command is a one way switch, in other words, it cannot be reverted (see <https://tex.stackexchange.com/a/444057>). So, even if the fact that it is a “switch” rather than an environment complicates things, because we have to make ungrouped settings to correspond to its effects, in practice this is not a big deal, since these settings are never really reverted (by default, at least). Hence, hooking into `\appendix` is a viable and natural alternative. The `appendix` package defines the `appendices` environment, which provides for a way for the appendix to “end”, but in this case, of course, we can hook into the environment instead.

FIXME But using the hook in a command we are not sure of the content is more risky than I had presumed (see <https://tex.stackexchange.com/q/617905>, thanks Phelype Oleinik). So, decide later whether I want to keep this, or if the appendix should just be turned into a “how to”.

```

2559 \AddToHook { cmd / appendix / before }
2560 {
2561     \zcsetup
2562     {
2563         countertype =
2564         {

```

```

2565         chapter      = appendix ,
2566         section       = appendix ,
2567         subsection    = appendix ,
2568         subsubsection = appendix ,
2569     }
2570 }
2571 }
2572 % \begin{macrocode}
2573 %
2574 %
2575 %
2576 % \subsection{\pkg{appendix} package}
2577 %
2578 % \begin{macrocode}
2579 \AddToHook { begindocument }
2580 {
2581     \@ifpackageloaded { appendix }
2582     {
2583         \AddToHook { env / appendices / begin }
2584         {
2585             \zcsetup
2586             {
2587                 countertype =
2588                 {
2589                     chapter      = appendix ,
2590                     section       = appendix ,
2591                     subsection    = appendix ,
2592                     subsubsection = appendix ,
2593                 }
2594             }
2595         }
2596         \AddToHook { env / subappendices / begin }
2597         {
2598             \zcsetup
2599             {
2600                 countertype =
2601                 {
2602                     chapter      = subappendix ,
2603                     section       = subappendix ,
2604                     subsection    = subappendix ,
2605                     subsubsection = subappendix ,
2606                 }
2607             }
2608         }
2609     }
2610     {}
2611 }

```

9.2 memoir class

```

2612 \AddToHook { begindocument }
2613 {
2614     \@ifclassloaded { memoir }
2615     {

```

```

2616 \AddToHook { env / appendices / begin }
2617 {
2618   \zcsetup
2619   {
2620     countertype =
2621     {
2622       chapter      = appendix ,
2623       section      = appendix ,
2624       subsection   = appendix ,
2625       subsubsection = appendix ,
2626     }
2627   }
2628 }
2629 \AddToHook { env / subappendices / begin }
2630 {
2631   \zcsetup
2632   {
2633     countertype =
2634     {
2635       chapter      = subappendix ,
2636       section      = subappendix ,
2637       subsection   = subappendix ,
2638       subsubsection = subappendix ,
2639     }
2640   }
2641 }
2642 }
2643 {}
2644 }

```

9.3 listings package

```

2645 \AddToHook { begindocument }
2646 {
2647   \@ifpackageloaded { listings }
2648   {
2649     \zcsetup
2650     {
2651       countertype =
2652       {
2653         lstlisting = listing ,
2654         lstnumber  = line ,
2655       } ,
2656       counterresetby = { lstnumber = lstlisting } ,
2657     }
2658     \lst@AddToHook { Init }
2659     {

```

Set (also) a `\zlabel` with the label received in the `label=` option from the `lstlisting` environment.

```

2660       \tl_if_empty:NF \lst@label
2661       { \zlabel { \lst@label } }

```

The correct place to set `currentcounter` to `lstnumber` is indeed the `Init` hook, since `listings` itself sets `\@currentlabel` to `\thelstnumber` in the same hook. See section “Line numbers” of ‘`texdoc listings-devel`’ (the `.dtx`), and search for the definition of

macro `\c@lstnumber`. Note that listings *does use* `\refstepcounter{lstnumber}`, but does so in the `EveryPar` hook, and there must be some grouping involved such that `\@currentcounter` ends up not being visible to the label. Indeed, the fact that listings manually sets `\@currentlabel` to `\thelstnumber` is a signal that the work of `\refstepcounter` is being restrained somehow.

```

2662         \zcsetup { currentcounter = lstnumber }
2663     }
2664 }
2665 {}
2666 }

```

9.4 enumitem package

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

```

2667 \</package>

```

10 Dictionaries

10.1 English

```

2668 <package>\zcDeclareLanguage { english }
2669 <package>\zcDeclareLanguageAlias { american } { english }
2670 <package>\zcDeclareLanguageAlias { australian } { english }
2671 <package>\zcDeclareLanguageAlias { british } { english }
2672 <package>\zcDeclareLanguageAlias { canadian } { english }
2673 <package>\zcDeclareLanguageAlias { newzealand } { english }
2674 <package>\zcDeclareLanguageAlias { UKenglish } { english }
2675 <package>\zcDeclareLanguageAlias { USenglish } { english }
2676 <*dict-english>
2677 namesep = {\nobreakspace} ,
2678 pairsep = {\and\nobreakspace} ,
2679 listsep = {,~} ,
2680 lastsep = {\and\nobreakspace} ,
2681 tpairsep = {\and\nobreakspace} ,
2682 tlistsep = {,~} ,
2683 tlastsep = {,~and\nobreakspace} ,
2684 notesep = {~} ,
2685 rangesep = {\to\nobreakspace} ,
2686
2687 type = part ,
2688   Name-sg = Part ,
2689   name-sg = part ,
2690   Name-pl = Parts ,
2691   name-pl = parts ,
2692
2693 type = chapter ,
2694   Name-sg = Chapter ,
2695   name-sg = chapter ,
2696   Name-pl = Chapters ,
2697   name-pl = chapters ,
2698

```

```

2699 type = section ,
2700     Name-sg = Section ,
2701     name-sg = section ,
2702     Name-pl = Sections ,
2703     name-pl = sections ,
2704
2705 type = paragraph ,
2706     Name-sg = Paragraph ,
2707     name-sg = paragraph ,
2708     Name-pl = Paragraphs ,
2709     name-pl = paragraphs ,
2710     Name-sg-ab = Par. ,
2711     name-sg-ab = par. ,
2712     Name-pl-ab = Par. ,
2713     name-pl-ab = par. ,
2714
2715 type = appendix ,
2716     Name-sg = Appendix ,
2717     name-sg = appendix ,
2718     Name-pl = Appendices ,
2719     name-pl = appendices ,
2720
2721 type = subappendix ,
2722     Name-sg = Appendix ,
2723     name-sg = appendix ,
2724     Name-pl = Appendices ,
2725     name-pl = appendices ,
2726
2727 type = page ,
2728     Name-sg = Page ,
2729     name-sg = page ,
2730     Name-pl = Pages ,
2731     name-pl = pages ,
2732     name-sg-ab = p. ,
2733     name-pl-ab = pp. ,
2734
2735 type = line ,
2736     Name-sg = Line ,
2737     name-sg = line ,
2738     Name-pl = Lines ,
2739     name-pl = lines ,
2740
2741 type = figure ,
2742     Name-sg = Figure ,
2743     name-sg = figure ,
2744     Name-pl = Figures ,
2745     name-pl = figures ,
2746     Name-sg-ab = Fig. ,
2747     name-sg-ab = fig. ,
2748     Name-pl-ab = Figs. ,
2749     name-pl-ab = figs. ,
2750
2751 type = table ,
2752     Name-sg = Table ,

```



```

2753     name-sg = table ,
2754     Name-pl = Tables ,
2755     name-pl = tables ,
2756
2757 type = item ,
2758     Name-sg = Item ,
2759     name-sg = item ,
2760     Name-pl = Items ,
2761     name-pl = items ,
2762
2763 type = footnote ,
2764     Name-sg = Footnote ,
2765     name-sg = footnote ,
2766     Name-pl = Footnotes ,
2767     name-pl = footnotes ,
2768
2769 type = note ,
2770     Name-sg = Note ,
2771     name-sg = note ,
2772     Name-pl = Notes ,
2773     name-pl = notes ,
2774
2775 type = equation ,
2776     Name-sg = Equation ,
2777     name-sg = equation ,
2778     Name-pl = Equations ,
2779     name-pl = equations ,
2780     Name-sg-ab = Eq. ,
2781     name-sg-ab = eq. ,
2782     Name-pl-ab = Eqs. ,
2783     name-pl-ab = eqs. ,
2784     refpre-in = {(} ,
2785     refpos-in = {)} ,
2786
2787 type = theorem ,
2788     Name-sg = Theorem ,
2789     name-sg = theorem ,
2790     Name-pl = Theorems ,
2791     name-pl = theorems ,
2792
2793 type = lemma ,
2794     Name-sg = Lemma ,
2795     name-sg = lemma ,
2796     Name-pl = Lemmas ,
2797     name-pl = lemmas ,
2798
2799 type = corollary ,
2800     Name-sg = Corollary ,
2801     name-sg = corollary ,
2802     Name-pl = Corollaries ,
2803     name-pl = corollaries ,
2804
2805 type = proposition ,
2806     Name-sg = Proposition ,

```

```

2807     name-sg = proposition ,
2808     Name-pl = Propositions ,
2809     name-pl = propositions ,
2810
2811 type = definition ,
2812     Name-sg = Definition ,
2813     name-sg = definition ,
2814     Name-pl = Definitions ,
2815     name-pl = definitions ,
2816
2817 type = proof ,
2818     Name-sg = Proof ,
2819     name-sg = proof ,
2820     Name-pl = Proofs ,
2821     name-pl = proofs ,
2822
2823 type = result ,
2824     Name-sg = Result ,
2825     name-sg = result ,
2826     Name-pl = Results ,
2827     name-pl = results ,
2828
2829 type = remark ,
2830     Name-sg = Remark ,
2831     name-sg = remark ,
2832     Name-pl = Remarks ,
2833     name-pl = remarks ,
2834
2835 type = example ,
2836     Name-sg = Example ,
2837     name-sg = example ,
2838     Name-pl = Examples ,
2839     name-pl = examples ,
2840
2841 type = algorithm ,
2842     Name-sg = Algorithm ,
2843     name-sg = algorithm ,
2844     Name-pl = Algorithms ,
2845     name-pl = algorithms ,
2846
2847 type = listing ,
2848     Name-sg = Listing ,
2849     name-sg = listing ,
2850     Name-pl = Listings ,
2851     name-pl = listings ,
2852
2853 type = exercise ,
2854     Name-sg = Exercise ,
2855     name-sg = exercise ,
2856     Name-pl = Exercises ,
2857     name-pl = exercises ,
2858
2859 type = solution ,
2860     Name-sg = Solution ,

```

```

2861 name-sg = solution ,
2862 Name-pl = Solutions ,
2863 name-pl = solutions ,
2864 </dict-english>

```

10.2 German

```

2865 <package>\zcDeclareLanguage { german }
2866 <package>\zcDeclareLanguageAlias { austrian      } { german }
2867 <package>\zcDeclareLanguageAlias { germanb       } { german }
2868 <package>\zcDeclareLanguageAlias { ngerman        } { german }
2869 <package>\zcDeclareLanguageAlias { naustrian      } { german }
2870 <package>\zcDeclareLanguageAlias { nswissgerman   } { german }
2871 <package>\zcDeclareLanguageAlias { swissgerman    } { german }
2872 <*dict-german>

2873 namesep = {\nobreakspace} ,
2874 pairsep  = {\~und\nobreakspace} ,
2875 listsep  = {,~} ,
2876 lastsep  = {\~und\nobreakspace} ,
2877 tpairsep = {\~und\nobreakspace} ,
2878 tlistsep = {,~} ,
2879 tlastsep = {\~und\nobreakspace} ,
2880 notesep  = {\~} ,
2881 rangesep = {\~bis\nobreakspace} ,
2882
2883 type = part ,
2884   Name-sg = Teil ,
2885   name-sg = Teil ,
2886   Name-pl = Teile ,
2887   name-pl = Teile ,
2888
2889 type = chapter ,
2890   Name-sg = Kapitel ,
2891   name-sg = Kapitel ,
2892   Name-pl = Kapitel ,
2893   name-pl = Kapitel ,
2894
2895 type = section ,
2896   Name-sg = Abschnitt ,
2897   name-sg = Abschnitt ,
2898   Name-pl = Abschnitte ,
2899   name-pl = Abschnitte ,
2900
2901 type = paragraph ,
2902   Name-sg = Absatz ,
2903   name-sg = Absatz ,
2904   Name-pl = Absätze ,
2905   name-pl = Absätze ,
2906
2907 type = appendix ,
2908   Name-sg = Anhang ,
2909   name-sg = Anhang ,
2910   Name-pl = Anhänge ,
2911   name-pl = Anhänge ,

```

```

2912
2913 type = subappendix ,
2914     Name-sg = Anhang ,
2915     name-sg = Anhang ,
2916     Name-pl = Anhänge ,
2917     name-pl = Anhänge ,
2918
2919 type = page ,
2920     Name-sg = Seite ,
2921     name-sg = Seite ,
2922     Name-pl = Seiten ,
2923     name-pl = Seiten ,
2924
2925 type = line ,
2926     Name-sg = Zeile ,
2927     name-sg = Zeile ,
2928     Name-pl = Zeilen ,
2929     name-pl = Zeilen ,
2930
2931 type = figure ,
2932     Name-sg = Abbildung ,
2933     name-sg = Abbildung ,
2934     Name-pl = Abbildungen ,
2935     name-pl = Abbildungen ,
2936     Name-sg-ab = Abb. ,
2937     name-sg-ab = Abb. ,
2938     Name-pl-ab = Abb. ,
2939     name-pl-ab = Abb. ,
2940
2941 type = table ,
2942     Name-sg = Tabelle ,
2943     name-sg = Tabelle ,
2944     Name-pl = Tabellen ,
2945     name-pl = Tabellen ,
2946
2947 type = item ,
2948     Name-sg = Punkt ,
2949     name-sg = Punkt ,
2950     Name-pl = Punkte ,
2951     name-pl = Punkte ,
2952
2953 type = footnote ,
2954     Name-sg = Fußnote ,
2955     name-sg = Fußnote ,
2956     Name-pl = Fußnoten ,
2957     name-pl = Fußnoten ,
2958
2959 type = note ,
2960     Name-sg = Anmerkung ,
2961     name-sg = Anmerkung ,
2962     Name-pl = Anmerkungen ,
2963     name-pl = Anmerkungen ,
2964
2965 type = equation ,

```

```

2966 Name-sg = Gleichung ,
2967 name-sg = Gleichung ,
2968 Name-pl = Gleichungen ,
2969 name-pl = Gleichungen ,
2970 refpre-in = {} ,
2971 refpos-in = {} ,
2972
2973 type = theorem ,
2974 Name-sg = Theorem ,
2975 name-sg = Theorem ,
2976 Name-pl = Theoreme ,
2977 name-pl = Theoreme ,
2978
2979 type = lemma ,
2980 Name-sg = Lemma ,
2981 name-sg = Lemma ,
2982 Name-pl = Lemmata ,
2983 name-pl = Lemmata ,
2984
2985 type = corollary ,
2986 Name-sg = Korollar ,
2987 name-sg = Korollar ,
2988 Name-pl = Korollare ,
2989 name-pl = Korollare ,
2990
2991 type = proposition ,
2992 Name-sg = Satz ,
2993 name-sg = Satz ,
2994 Name-pl = Sätze ,
2995 name-pl = Sätze ,
2996
2997 type = definition ,
2998 Name-sg = Definition ,
2999 name-sg = Definition ,
3000 Name-pl = Definitionen ,
3001 name-pl = Definitionen ,
3002
3003 type = proof ,
3004 Name-sg = Beweis ,
3005 name-sg = Beweis ,
3006 Name-pl = Beweise ,
3007 name-pl = Beweise ,
3008
3009 type = result ,
3010 Name-sg = Ergebnis ,
3011 name-sg = Ergebnis ,
3012 Name-pl = Ergebnisse ,
3013 name-pl = Ergebnisse ,
3014
3015 type = remark ,
3016 Name-sg = Bemerkung ,
3017 name-sg = Bemerkung ,
3018 Name-pl = Bemerkungen ,
3019 name-pl = Bemerkungen ,

```

```

3020
3021 type = example ,
3022   Name-sg = Beispiel ,
3023   name-sg = Beispiel ,
3024   Name-pl = Beispiele ,
3025   name-pl = Beispiele ,
3026
3027 type = algorithm ,
3028   Name-sg = Algorithmus ,
3029   name-sg = Algorithmus ,
3030   Name-pl = Algorithmen ,
3031   name-pl = Algorithmen ,
3032
3033 type = listing ,
3034   Name-sg = Listing ,
3035   name-sg = Listing ,
3036   Name-pl = Listings ,
3037   name-pl = Listings ,
3038
3039 type = exercise ,
3040   Name-sg = Übungsaufgabe ,
3041   name-sg = Übungsaufgabe ,
3042   Name-pl = Übungsaufgaben ,
3043   name-pl = Übungsaufgaben ,
3044
3045 type = solution ,
3046   Name-sg = Lösung ,
3047   name-sg = Lösung ,
3048   Name-pl = Lösungen ,
3049   name-pl = Lösungen ,
3050 </dict-german>

```

10.3 French

```

3051 <package>\zcDeclareLanguage { french }
3052 <package>\zcDeclareLanguageAlias { acadian } { french }
3053 <package>\zcDeclareLanguageAlias { canadien } { french }
3054 <package>\zcDeclareLanguageAlias { francais } { french }
3055 <package>\zcDeclareLanguageAlias { frenchb } { french }
3056 <*dict-french>
3057 namesep = {\nobreakspace} ,
3058 pairsep = {\~et\nobreakspace} ,
3059 listsep = {,~} ,
3060 lastsep = {\~et\nobreakspace} ,
3061 tpairsep = {\~et\nobreakspace} ,
3062 tlistsep = {,~} ,
3063 tlastsep = {\~et\nobreakspace} ,
3064 notesep = {\~} ,
3065 rangesep = {\~à\nobreakspace} ,
3066
3067 type = part ,
3068   Name-sg = Partie ,
3069   name-sg = partie ,
3070   Name-pl = Parties ,

```

```

3071     name-pl = parties ,
3072
3073 type = chapter ,
3074     Name-sg = Chapitre ,
3075     name-sg = chapitre ,
3076     Name-pl = Chapitres ,
3077     name-pl = chapitres ,
3078
3079 type = section ,
3080     Name-sg = Section ,
3081     name-sg = section ,
3082     Name-pl = Sections ,
3083     name-pl = sections ,
3084
3085 type = paragraph ,
3086     Name-sg = Paragraphe ,
3087     name-sg = paragraphe ,
3088     Name-pl = Paragraphes ,
3089     name-pl = paragraphes ,
3090
3091 type = appendix ,
3092     Name-sg = Annexe ,
3093     name-sg = annexe ,
3094     Name-pl = Annexes ,
3095     name-pl = annexes ,
3096
3097 type = subappendix ,
3098     Name-sg = Annexe ,
3099     name-sg = annexe ,
3100     Name-pl = Annexes ,
3101     name-pl = annexes ,
3102
3103 type = page ,
3104     Name-sg = Page ,
3105     name-sg = page ,
3106     Name-pl = Pages ,
3107     name-pl = pages ,
3108
3109 type = line ,
3110     Name-sg = Ligne ,
3111     name-sg = ligne ,
3112     Name-pl = Lignes ,
3113     name-pl = lignes ,
3114
3115 type = figure ,
3116     Name-sg = Figure ,
3117     name-sg = figure ,
3118     Name-pl = Figures ,
3119     name-pl = figures ,
3120
3121 type = table ,
3122     Name-sg = Table ,
3123     name-sg = table ,
3124     Name-pl = Tables ,

```

```

3125     name-pl = tables ,
3126
3127 type = item ,
3128     Name-sg = Point ,
3129     name-sg = point ,
3130     Name-pl = Points ,
3131     name-pl = points ,
3132
3133 type = footnote ,
3134     Name-sg = Note ,
3135     name-sg = note ,
3136     Name-pl = Notes ,
3137     name-pl = notes ,
3138
3139 type = note ,
3140     Name-sg = Note ,
3141     name-sg = note ,
3142     Name-pl = Notes ,
3143     name-pl = notes ,
3144
3145 type = equation ,
3146     Name-sg = Équation ,
3147     name-sg = équation ,
3148     Name-pl = Équations ,
3149     name-pl = équations ,
3150     refpre-in = {() ,
3151     refpos-in = {} } ,
3152
3153 type = theorem ,
3154     Name-sg = Théorème ,
3155     name-sg = théorème ,
3156     Name-pl = Théorèmes ,
3157     name-pl = théorèmes ,
3158
3159 type = lemma ,
3160     Name-sg = Lemme ,
3161     name-sg = lemme ,
3162     Name-pl = Lemmes ,
3163     name-pl = lemmes ,
3164
3165 type = corollary ,
3166     Name-sg = Corollaire ,
3167     name-sg = corollaire ,
3168     Name-pl = Corollaires ,
3169     name-pl = corollaires ,
3170
3171 type = proposition ,
3172     Name-sg = Proposition ,
3173     name-sg = proposition ,
3174     Name-pl = Propositions ,
3175     name-pl = propositions ,
3176
3177 type = definition ,
3178     Name-sg = Définition ,

```



```

3179     name-sg = définition ,
3180     Name-pl = Définitions ,
3181     name-pl = définitions ,
3182
3183 type = proof ,
3184     Name-sg = Démonstration ,
3185     name-sg = démonstration ,
3186     Name-pl = Démonstrations ,
3187     name-pl = démonstrations ,
3188
3189 type = result ,
3190     Name-sg = Résultat ,
3191     name-sg = résultat ,
3192     Name-pl = Résultats ,
3193     name-pl = résultats ,
3194
3195 type = remark ,
3196     Name-sg = Remarque ,
3197     name-sg = remarque ,
3198     Name-pl = Remarques ,
3199     name-pl = remarques ,
3200
3201 type = example ,
3202     Name-sg = Exemple ,
3203     name-sg = exemple ,
3204     Name-pl = Exemples ,
3205     name-pl = exemples ,
3206
3207 type = algorithm ,
3208     Name-sg = Algorithme ,
3209     name-sg = algorithme ,
3210     Name-pl = Algorithmes ,
3211     name-pl = algorithmes ,
3212
3213 type = listing ,
3214     Name-sg = Liste ,
3215     name-sg = liste ,
3216     Name-pl = Listes ,
3217     name-pl = listes ,
3218
3219 type = exercise ,
3220     Name-sg = Exercice ,
3221     name-sg = exercice ,
3222     Name-pl = Exercices ,
3223     name-pl = exercices ,
3224
3225 type = solution ,
3226     Name-sg = Solution ,
3227     name-sg = solution ,
3228     Name-pl = Solutions ,
3229     name-pl = solutions ,
3230 </dict-french>

```

10.4 Portuguese

```
3231 <package>\zcDeclareLanguage { portuguese }
3232 <package>\zcDeclareLanguageAlias { brazilian } { portuguese }
3233 <package>\zcDeclareLanguageAlias { brazil    } { portuguese }
3234 <package>\zcDeclareLanguageAlias { portuges  } { portuguese }
3235 <*dict-portuguese>

3236 namesep = {\nobreakspace} ,
3237 pairsep  = {\~e\nobreakspace} ,
3238 listsep  = {,~} ,
3239 lastsep  = {\~e\nobreakspace} ,
3240 tpairsep = {\~e\nobreakspace} ,
3241 tlistsep = {,~} ,
3242 tlastsep = {\~e\nobreakspace} ,
3243 notesep  = {\~} ,
3244 rangesep = {\~a\nobreakspace} ,
3245
3246 type = part ,
3247   Name-sg = Parte ,
3248   name-sg = parte ,
3249   Name-pl  = Partes ,
3250   name-pl  = partes ,
3251
3252 type = chapter ,
3253   Name-sg = Capítulo ,
3254   name-sg = capítulo ,
3255   Name-pl  = Capítulos ,
3256   name-pl  = capítulos ,
3257
3258 type = section ,
3259   Name-sg = Seção ,
3260   name-sg = seção ,
3261   Name-pl  = Seções ,
3262   name-pl  = seções ,
3263
3264 type = paragraph ,
3265   Name-sg = Parágrafo ,
3266   name-sg = parágrafo ,
3267   Name-pl  = Parágrafos ,
3268   name-pl  = parágrafos ,
3269   Name-sg-ab = Par. ,
3270   name-sg-ab = par. ,
3271   Name-pl-ab = Par. ,
3272   name-pl-ab = par. ,
3273
3274 type = appendix ,
3275   Name-sg = Apêndice ,
3276   name-sg = apêndice ,
3277   Name-pl  = Apêndices ,
3278   name-pl  = apêndices ,
3279
3280 type = subappendix ,
3281   Name-sg = Apêndice ,
3282   name-sg = apêndice ,
```

```

3283     Name-pl = Apêndices ,
3284     name-pl = apêndices ,
3285
3286     type = page ,
3287     Name-sg = Página ,
3288     name-sg = página ,
3289     Name-pl = Páginas ,
3290     name-pl = páginas ,
3291     name-sg-ab = p. ,
3292     name-pl-ab = pp. ,
3293
3294     type = line ,
3295     Name-sg = Linha ,
3296     name-sg = linha ,
3297     Name-pl = Linhas ,
3298     name-pl = linhas ,
3299
3300     type = figure ,
3301     Name-sg = Figura ,
3302     name-sg = figura ,
3303     Name-pl = Figuras ,
3304     name-pl = figuras ,
3305     Name-sg-ab = Fig. ,
3306     name-sg-ab = fig. ,
3307     Name-pl-ab = Figs. ,
3308     name-pl-ab = figs. ,
3309
3310     type = table ,
3311     Name-sg = Tabela ,
3312     name-sg = tabela ,
3313     Name-pl = Tabelas ,
3314     name-pl = tabelas ,
3315
3316     type = item ,
3317     Name-sg = Item ,
3318     name-sg = item ,
3319     Name-pl = Itens ,
3320     name-pl = itens ,
3321
3322     type = footnote ,
3323     Name-sg = Nota ,
3324     name-sg = nota ,
3325     Name-pl = Notas ,
3326     name-pl = notas ,
3327
3328     type = note ,
3329     Name-sg = Nota ,
3330     name-sg = nota ,
3331     Name-pl = Notas ,
3332     name-pl = notas ,
3333
3334     type = equation ,
3335     Name-sg = Equação ,
3336     name-sg = equação ,

```

```

3337 Name-pl = Equações ,
3338 name-pl = equações ,
3339 Name-sg-ab = Eq. ,
3340 name-sg-ab = eq. ,
3341 Name-pl-ab = Eqs. ,
3342 name-pl-ab = eqs. ,
3343 refpre-in = {()} ,
3344 refpos-in = {} ,
3345
3346 type = theorem ,
3347 Name-sg = Teorema ,
3348 name-sg = teorema ,
3349 Name-pl = Teoremas ,
3350 name-pl = teoremas ,
3351
3352 type = lemma ,
3353 Name-sg = Lema ,
3354 name-sg = lema ,
3355 Name-pl = Lemas ,
3356 name-pl = lemas ,
3357
3358 type = corollary ,
3359 Name-sg = Corolário ,
3360 name-sg = corolário ,
3361 Name-pl = Corolários ,
3362 name-pl = corolários ,
3363
3364 type = proposition ,
3365 Name-sg = Proposição ,
3366 name-sg = proposição ,
3367 Name-pl = Proposições ,
3368 name-pl = proposições ,
3369
3370 type = definition ,
3371 Name-sg = Definição ,
3372 name-sg = definição ,
3373 Name-pl = Definições ,
3374 name-pl = definições ,
3375
3376 type = proof ,
3377 Name-sg = Demonstração ,
3378 name-sg = demonstração ,
3379 Name-pl = Demonstrações ,
3380 name-pl = demonstrações ,
3381
3382 type = result ,
3383 Name-sg = Resultado ,
3384 name-sg = resultado ,
3385 Name-pl = Resultados ,
3386 name-pl = resultados ,
3387
3388 type = remark ,
3389 Name-sg = Observação ,
3390 name-sg = observação ,

```

```

3391   Name-pl = Observações ,
3392   name-pl = observações ,
3393
3394   type = example ,
3395   Name-sg = Exemplo ,
3396   name-sg = exemplo ,
3397   Name-pl = Exemplos ,
3398   name-pl = exemplos ,
3399
3400   type = algorithm ,
3401   Name-sg = Algoritmo ,
3402   name-sg = algoritmo ,
3403   Name-pl = Algoritmos ,
3404   name-pl = algoritmos ,
3405
3406   type = listing ,
3407   Name-sg = Listagem ,
3408   name-sg = listagem ,
3409   Name-pl = Listagens ,
3410   name-pl = listagens ,
3411
3412   type = exercise ,
3413   Name-sg = Exercício ,
3414   name-sg = exercício ,
3415   Name-pl = Exercícios ,
3416   name-pl = exercícios ,
3417
3418   type = solution ,
3419   Name-sg = Solução ,
3420   name-sg = solução ,
3421   Name-pl = Soluções ,
3422   name-pl = soluções ,
3423 </dict-portuguese>

```

10.5 Spanish

```

3424 <package>\zcDeclareLanguage { spanish }
3425 <*dict-spanish>
3426 namesep = {\nobreakspace} ,
3427 pairsep = {\sim\nobreakspace} ,
3428 listsep = {,~} ,
3429 lastsep = {\sim\nobreakspace} ,
3430 tpairsep = {\sim\nobreakspace} ,
3431 tlistsep = {,~} ,
3432 tlastsep = {\sim\nobreakspace} ,
3433 notesep = {\sim} ,
3434 rangesep = {\sim a\nobreakspace} ,
3435
3436 type = part ,
3437 Name-sg = Parte ,
3438 name-sg = parte ,
3439 Name-pl = Partes ,
3440 name-pl = partes ,
3441

```

```

3442 type = chapter ,
3443     Name-sg = Capítulo ,
3444     name-sg = capítulo ,
3445     Name-pl = Capítulos ,
3446     name-pl = capítulos ,
3447
3448 type = section ,
3449     Name-sg = Sección ,
3450     name-sg = sección ,
3451     Name-pl = Secciones ,
3452     name-pl = secciones ,
3453
3454 type = paragraph ,
3455     Name-sg = Párrafo ,
3456     name-sg = párrafo ,
3457     Name-pl = Párrafos ,
3458     name-pl = párrafos ,
3459
3460 type = appendix ,
3461     Name-sg = Apéndice ,
3462     name-sg = apéndice ,
3463     Name-pl = Apéndices ,
3464     name-pl = apéndices ,
3465
3466 type = subappendix ,
3467     Name-sg = Apéndice ,
3468     name-sg = apéndice ,
3469     Name-pl = Apéndices ,
3470     name-pl = apéndices ,
3471
3472 type = page ,
3473     Name-sg = Página ,
3474     name-sg = página ,
3475     Name-pl = Páginas ,
3476     name-pl = páginas ,
3477
3478 type = line ,
3479     Name-sg = Línea ,
3480     name-sg = línea ,
3481     Name-pl = Líneas ,
3482     name-pl = líneas ,
3483
3484 type = figure ,
3485     Name-sg = Figura ,
3486     name-sg = figura ,
3487     Name-pl = Figuras ,
3488     name-pl = figuras ,
3489
3490 type = table ,
3491     Name-sg = Cuadro ,
3492     name-sg = cuadro ,
3493     Name-pl = Cuadros ,
3494     name-pl = cuadros ,
3495

```

```

3496 type = item ,
3497     Name-sg = Punto ,
3498     name-sg = punto ,
3499     Name-pl = Puntos ,
3500     name-pl = puntos ,
3501
3502 type = footnote ,
3503     Name-sg = Nota ,
3504     name-sg = nota ,
3505     Name-pl = Notas ,
3506     name-pl = notas ,
3507
3508 type = note ,
3509     Name-sg = Nota ,
3510     name-sg = nota ,
3511     Name-pl = Notas ,
3512     name-pl = notas ,
3513
3514 type = equation ,
3515     Name-sg = Ecuación ,
3516     name-sg = ecuación ,
3517     Name-pl = Ecuaciones ,
3518     name-pl = ecuaciones ,
3519     refpre-in = {()} ,
3520     refpos-in = {} ,
3521
3522 type = theorem ,
3523     Name-sg = Teorema ,
3524     name-sg = teorema ,
3525     Name-pl = Teoremas ,
3526     name-pl = teoremas ,
3527
3528 type = lemma ,
3529     Name-sg = Lema ,
3530     name-sg = lema ,
3531     Name-pl = Lemas ,
3532     name-pl = lemas ,
3533
3534 type = corollary ,
3535     Name-sg = Corolario ,
3536     name-sg = corolario ,
3537     Name-pl = Corolarios ,
3538     name-pl = corolarios ,
3539
3540 type = proposition ,
3541     Name-sg = Proposición ,
3542     name-sg = proposición ,
3543     Name-pl = Propositiones ,
3544     name-pl = proposiciones ,
3545
3546 type = definition ,
3547     Name-sg = Definición ,
3548     name-sg = definición ,
3549     Name-pl = Definiciones ,

```

```

3550     name-pl = definiciones ,
3551
3552 type = proof ,
3553     Name-sg = Demostración ,
3554     name-sg = demostración ,
3555     Name-pl = Demostraciones ,
3556     name-pl = demostraciones ,
3557
3558 type = result ,
3559     Name-sg = Resultado ,
3560     name-sg = resultado ,
3561     Name-pl = Resultados ,
3562     name-pl = resultados ,
3563
3564 type = remark ,
3565     Name-sg = Observación ,
3566     name-sg = observación ,
3567     Name-pl = Observaciones ,
3568     name-pl = observaciones ,
3569
3570 type = example ,
3571     Name-sg = Ejemplo ,
3572     name-sg = ejemplo ,
3573     Name-pl = Ejemplos ,
3574     name-pl = ejemplos ,
3575
3576 type = algorithm ,
3577     Name-sg = Algoritmo ,
3578     name-sg = algoritmo ,
3579     Name-pl = Algoritmos ,
3580     name-pl = algoritmos ,
3581
3582 type = listing ,
3583     Name-sg = Listado ,
3584     name-sg = listado ,
3585     Name-pl = Listados ,
3586     name-pl = listados ,
3587
3588 type = exercise ,
3589     Name-sg = Ejercicio ,
3590     name-sg = ejercicio ,
3591     Name-pl = Ejercicios ,
3592     name-pl = ejercicios ,
3593
3594 type = solution ,
3595     Name-sg = Solución ,
3596     name-sg = solución ,
3597     Name-pl = Soluciones ,
3598     name-pl = soluciones ,
3599 </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 | | C | |
|----------------------------------|--|-------------------------|--|
| \\ | 107, 113, 122, 123, 128, 129, 134, 135, 144, 145, 155 | clist commands: | |
| † internal commands: | | \clist_map_inline:nn | 842 |
| \z_zrefclever_current_counter_tl | 4 | \counterwithin | 4 |
| | | cs commands: | |
| | | \cs_generate_variant:Nn | 59, 60, 314, 322, 985, 991, 1234, 2142 |
| | | \cs_if_exist:NTF | 43, 52, 73 |
| | | \cs_new:Npn | 41, 50, 61, 71, 82, 2093, 2143 |
| | | \cs_new_protected:Npn | 262, 315, 323, 329, 450, 980, 986, 1069, 1122, 1164, 1175, 1235, 1407, 1459, 1503, 1665, 1933, 2089, 2091, 2295, 2422, 2475, 2532 |
| | | \cs_new_protected:Npx | 94 |
| | | \cs_set_eq:NN | 98 |
| A | | E | |
| \AddToHook | 95, 466, 481, 625, 661, 686, 724, 726, 786, 2559, 2579, 2583, 2596, 2612, 2616, 2629, 2645 | \endinput | 12 |
| \appendix | 68 | exp commands: | |
| \appendixname | 68 | \exp_args:NNe | 27, 30 |
| | | \exp_args:NNnx | 252 |
| | | \exp_args:NnV | 290 |
| | | \exp_args:NNx | 99 |
| | | \exp_args:Nnx | 325, 1295, 1332 |
| | | \exp_args:Nx | 272 |
| | | \exp_args:Nxx | 1218, 1265, 1368, 2426, 2448, 2452 |
| | | \exp_not:N | 58, 1843, 1846, 1868, 1871, 1874, 2103, 2106, 2109, 2122, 2124, 2127, 2130, 2135, 2137, 2155, 2176, 2179, 2181, 2184, 2191, 2198, 2200, 2204, 2207, 2210, 2222, 2225, 2238, 2241, 2244, 2271, 2273, 2276, 2279, 2286, 2288 |
| | | \exp_not:n | 1687, 1703, 1715, 1720, 1743, 1757, 1761, 1773, 1777, 1811, 1812, 1844, 1867, 1872, 1873, 2002, 2015, 2022, 2046, 2058, 2062, 2072, 2076, 2104, 2105, 2107, 2117, 2120, 2123, 2128, 2129, 2131, 2132, 2134, 2136, 2177, 2178, 2180, 2182, 2183, 2185, 2186, 2190, 2202, 2203, 2208, 2209, 2211, 2219, 2223, 2224, 2226, 2239, 2240, 2242, 2265, 2269, 2272, 2277, 2278, 2280, 2281, 2285, 2287 |
| | | \ExplSyntaxOn | 12, 274 |
| B | | | |
| \babelname | 671 | | |
| \babelprovide | 11, 22 | | |
| \begin | 2572, 2578 | | |
| bool commands: | | | |
| \bool_case_true: | 2 | | |
| \bool_if:NTF | 298, 309, 629, 633, 1313, 1355, 1563, 1658, 1788, 1810, 1841, 1898, 1939, 1962, 1966, 1972, 1982, 1988, 2148 | | |
| \bool_if:nTF | 63, 1182, 1191, 1200, 1257, 1285, 1322, 1429, 1437, 1577, 1585, 1822, 1829, 1836, 2097, 2232 | | |
| \bool_lazy_all:nTF | 2481 | | |
| \bool_lazy_and:nnTF | 1076, 1094, 2317, 2538 | | |
| \bool_lazy_any:nTF | 2395, 2404 | | |
| \bool_lazy_or:nnTF | 1080, 2305 | | |
| \bool_new:N | 261, 502, 503, 528, 552, 561, 568, 569, 582, 583, 602, 603, 779, 780, 1105, 1118, 1469, 1470, 1480, 1486, 1487 | | |
| \bool_set:Nn | 1074 | | |
| \bool_set_false:N | 515, 519, 610, 619, 620, 635, 801, 1254, 1526, 1569, 1583, 1597, 1800, 1937, 1938, 2402, 2419 | | |
| \bool_set_true:N | 318, 509, 510, 514, 520, 609, 614, 615, 790, 795, 1269, 1280, 1309, 1317, 1350, 1359, 1387, 1399, 1534, 1564, 1570, 1574, 1601, 1607, 2418, 2434, 2441, 2442, 2460, 2467, 2468 | | |
| \bool_until_do:Nn | 1255, 1527 | | |

| | | |
|--|--|--|
| F | | keyval commands: |
| file commands: | | \keyval_parse:nnn 814, 868 |
| \file_get:nnNTF 272 | | |
| \fmtversion 3 | | |
| G | | L |
| group commands: | | \labelformat 3 |
| \group_begin: .. 97, 264, 317, 969, | | \language name 22, 665 |
| 1071, 1084, 1843, 1871, 2103, 2106, | | |
| 2127, 2130, 2176, 2181, 2184, 2200, | | |
| 2207, 2222, 2238, 2241, 2276, 2279 | | |
| \group_end: 100, 312, 320, 977, | | M |
| 1087, 1102, 1868, 1874, 2122, 2124, | | \mainbabelname 22, 672 |
| 2135, 2137, 2179, 2191, 2198, 2204, | | \MessageBreak 10 |
| 2210, 2225, 2271, 2273, 2286, 2288 | | msg commands: |
| | | \msg_info:nnn 355, 385 |
| | | \msg_line_context: |
| | | 106, 112, 116, 133, 140, |
| | | 143, 149, 163, 167, 169, 171, 174, 178 |
| | | \msg_new:nnn 104, 110, 115, 117, 119, |
| | | 125, 131, 137, 139, 141, 147, 152, |
| | | 157, 159, 161, 166, 168, 170, 172, 177 |
| | | \msg_note:nnn 294 |
| | | \msg_warning:nn |
| | | 471, 496, 634, 640, 784, 805 |
| | | \msg_warning:nnn ... 241, 256, 300, |
| | | 310, 712, 757, 870, 934, 976, 1016, |
| | | 1055, 1622, 1795, 2348, 2383, 2524 |
| | | \msg_warning:nnnn |
| | | 816, 1278, 1315, 1357, 1397 |
| I | | N |
| \IfBooleanTF 1108 | | \newcounter 4 |
| \IfFormatAtLeastTF 3, 4 | | \NewDocumentCommand |
| \input 11, 12 | | .. 236, 246, 918, 920, 967, 1067, 1106 |
| int commands: | | \nobreakspace 399, |
| \int_case:nnTF | | 2677, 2678, 2680, 2681, 2683, 2685, |
| .. 1668, 1696, 1728, 1901, 1995, 2034 | | 2873, 2874, 2876, 2877, 2879, 2881, |
| \int_compare:nNnTF 1222, 1270, 1300, | | 3057, 3058, 3060, 3061, 3063, 3065, |
| 1337, 1372, 1388, 1415, 1417, 1461, | | 3236, 3237, 3239, 3240, 3242, 3244, |
| 1629, 1683, 1717, 1890, 1892, 1950, | | 3426, 3427, 3429, 3430, 3432, 3434 |
| 1975, 2019, 2430, 2436, 2456, 2462 | | |
| \int_compare_p:nNn | | P |
| 1431, 1439, 2309, 2320, 2415 | | \PackageError 7 |
| \int_eval:n 94 | | \pagenumbering 6 |
| \int_incr:N 1294, 1331, 1928, 1965, | | \pageref 35 |
| 1967, 1981, 1983, 1987, 1989, 2087 | | \pkg 2576 |
| \int_new:N | | prg commands: |
| .. 1119, 1120, 1471, 1472, 1483, 1484 | | \prg_generate_conditional_- |
| \int_set:Nn ... 1416, 1418, 1422, 1425 | | variant:Nnn 423, 439 |
| \int_use:N 37, 39, 54 | | \prg_new_protected_conditional:Npnn |
| \int_zero:N 1291, | | 409, 425, 442 |
| 1328, 1409, 1410, 1512, 1513, 1514, | | \prg_return_false: |
| 1515, 1927, 1929, 1930, 2082, 2083 | | 419, 421, 435, 437, 448 |
| \l_tmpa_int 1328, 1331, 1341 | | \prg_return_true: 418, 434, 447 |
| \l_tmpb_int 1291, 1294, 1305 | | \ProcessKeysOptions 917 |
| iow commands: | | prop commands: |
| \iow_char:N 107, 113, 122, | | \prop_get:NnN 2501 |
| 123, 128, 129, 134, 135, 144, 145, 155 | | |
| K | | |
| keys commands: | | |
| \keys_define:nn 30, 335, 347, | | |
| 364, 378, 457, 485, 492, 504, 529, | | |
| 538, 553, 562, 570, 584, 596, 604, | | |
| 637, 644, 682, 729, 771, 774, 781, | | |
| 791, 802, 810, 838, 864, 888, 898, | | |
| 909, 930, 942, 992, 1004, 1025, 1048 | | |
| \keys_set:nn 12, | | |
| 30, 34, 291, 796, 919, 925, 974, 1072 | | |

| | |
|--|---|
| \ZREF@mainlist | \tl_reverse_items:n |
| 21, 24, 35, 38, 40, 91, 92, 103 | 1234, 1240, 1244, 1248, 1252 |
| \zref@newprop | \tl_set:Nn |
| .. 5, 7, 20, 22, 25, 36, 39, 87, 89, 102 | 463, 469, 472, 488, 497, 665, 666, |
| \zref@refused | 671, 672, 675, 676, 679, 692, 700, |
| 1616 | 709, 714, 737, 745, 754, 759, 924, |
| \zref@wrapper@babel | 998, 1166, 1177, 1179, 1237, 1239, |
| 33, 1068 | 1241, 1243, 1245, 1247, 1249, 1251, |
| \textendash | 1377, 1379, 1381, 1383, 1543, 1544, |
| 403 | 1547, 1552, 1674, 1676, 1808, 1839, |
| \the | 1954, 1956, 1979, 2311, 2312, 2325 |
| 3 | \tl_set_eq:NN |
| \thechapter | 1921 |
| 68 | \tl_tail:N |
| \thelstnumber | 1378, 1380, 1382, 1384 |
| 70, 71 | \l_tmpa_tl |
| \thepage | 275, 291, 1090, 1091 |
| 6, 99 | |
| \thesection | |
| 68 | |
| tl commands: | |
| \c_empty_tl | |
| 1167, 1178, 1180, | |
| 1238, 1242, 1246, 1250, 1550, 1555 | |
| \c_novalue_tl | |
| 900, 944 | |
| \tl_clear:N | |
| 289, 340, 973, 997, 1507, | |
| 1508, 1509, 1510, 1511, 1533, 1923, | |
| 1924, 1925, 1926, 1964, 2298, 2301, | |
| 2329, 2347, 2382, 2523, 2554, 2556 | |
| \tl_gset:Nn | |
| 99 | |
| \tl_head:N | |
| .. 1369, 1370, 1373, 1375, 1389, 1391 | |
| \tl_if_empty:NNTF | |
| 75, 352, | |
| 369, 383, 1009, 1030, 1053, 1088, | |
| 1620, 1790, 2216, 2314, 2331, 2660 | |
| \tl_if_empty:nTF | |
| 238, 248, 339, 452, 996, 1739, 1755, | |
| 1771, 2013, 2044, 2056, 2070, 2300 | |
| \tl_if_empty_p:N | |
| 1186, 1187, 1195, | |
| 1196, 1203, 1204, 1580, 1581, 1588, | |
| 1590, 2399, 2409, 2413, 2483, 2539 | |
| \tl_if_empty_p:n | |
| 1261, 1262, 1288, 1325 | |
| \tl_if_eq:NNTF | |
| 1207, 1593 | |
| \tl_if_eq:NnTF | |
| 1125, 1157, | |
| 1421, 1424, 1449, 1452, 1541, 2424 | |
| \tl_if_eq:nnTF | |
| 1218, 1265, 1295, | |
| 1332, 1368, 1413, 2426, 2448, 2452 | |
| \tl_if_novalue:nTF | |
| 903, 947 | |
| \tl_item:Nn | |
| 1304, 1339 | |
| \tl_map_break:n | |
| 85, 1298, 1335 | |
| \tl_map_inline:Nn | |
| 1292, 1329 | |
| \tl_map_tokens:Nn | |
| 77 | |
| \tl_new:N | |
| 93, 179, 180, 456, | |
| 658, 659, 660, 770, 773, 887, 1112, | |
| 1113, 1114, 1115, 1116, 1117, 1473, | |
| 1474, 1475, 1476, 1477, 1478, 1479, | |
| 1481, 1482, 1485, 1488, 1489, 1490, | |
| 1491, 1492, 1493, 1494, 1495, 1496, | |
| 1497, 1498, 1499, 1500, 1501, 1502 | |
| \tl_put_left:Nn | |
| 1825, 1832, 1883 | |
| \tl_put_right:Nn | |
| 1685, 1701, | |
| 1710, 1741, 1752, 1768, 2000, 2011, | |
| 2042, 2054, 2068, 2315, 2316, 2327 | |
| | |
| | U |
| | use commands: |
| | \use:N |
| | 23 |
| | |
| | Z |
| | \zcDeclareLanguage |
| | 10, 236, 2668, 2865, 3051, 3231, 3424 |
| | \zcDeclareLanguageAlias |
| | 11, 246, 2669, 2670, |
| | 2671, 2672, 2673, 2674, 2675, 2866, |
| | 2867, 2868, 2869, 2870, 2871, 3052, |
| | 3053, 3054, 3055, 3232, 3233, 3234 |
| | \zcLanguageSetup |
| | 9, 11–13, 29, 31, 32, 967 |
| | \zcpageref |
| | 35, 1106 |
| | \zceref |
| | 24, 25, 28, |
| | 29, 33, 35–37, 45, 46, 1067, 1109, 1110 |
| | \zcRefTypeSetup |
| | 9, 29, 30, 920 |
| | \zcsetup |
| | 22, 25, 28, 29, 918, 2561, |
| | 2585, 2598, 2618, 2631, 2649, 2662 |
| | \zlabel |
| | 70, 2661 |
| | zrefcheck commands: |
| | \zrefcheck_zceref_beg_label: .. |
| | 1079 |
| | \zrefcheck_zceref_end_label_- |
| | maybe: |
| | 1098 |
| | \zrefcheck_zceref_run_checks_on_- |
| | labels:n |
| | 1099 |
| | zrefclever internal commands: |
| | \l__zrefclever_abbrev_bool |
| | 582, 586, 2318 |
| | \l__zrefclever_capitalize_bool .. |
| | 568, 572, 2306 |
| | \l__zrefclever_capitalize_first_- |
| | bool |
| | 569, 578, 2308 |
| | __zrefclever_counter_reset_by:n |
| | |
| | 5, 27, 43, 45, 47, 52, 54, 56, 61 |
| | __zrefclever_counter_reset_by_- |
| | aux:nn |
| | 68, 71 |
| | __zrefclever_counter_reset_by_- |
| | aux:nnn |
| | 78, 82 |

`\l_zrefclever_counter_resetby_`
`prop` 5, 27, 64, 65, 863, 875
`\l_zrefclever_counter_resettters_`
`seq` 4, 5, 27, 67, 837, 844, 847
`\l_zrefclever_counter_type_prop`
..... 3, 26, 27, 30, 809, 821
`\l_zrefclever_current_counter_`
`tl` 3, 28,
20, 23, 28, 31, 33, 37, 88, 90, 887, 890
`\l_zrefclever_current_language_`
`tl` .. 22, 660, 665, 671, 675, 701, 746
`_zrefclever_declare_default_`
`transl:nnn` ... 31, 980, 1011, 1032
`_zrefclever_declare_type_`
`transl:nnnn` ... 31, 980, 1037, 1059
`\g_zrefclever_dict_⟨language⟩_prop`
..... 12
`\l_zrefclever_dict_language_tl` .
. 179, 266, 270, 273, 280, 286, 293,
295, 301, 304, 326, 332, 413, 416,
429, 432, 971, 1012, 1033, 1038, 1060
`\g_zrefclever_fallback_dict_`
`prop` 9, 392, 393, 445
`_zrefclever_get_default_`
`transl:nnN` 9, 426, 440
`_zrefclever_get_default_`
`transl:nnNTF` 16, 425, 2516
`_zrefclever_get_enclosing_`
`counters:n` 5, 41, 46, 88
`_zrefclever_get_enclosing_`
`counters_value:n` ... 5, 41, 55, 90
`_zrefclever_get_fallback_`
`transl:nN` 443
`_zrefclever_get_fallback_`
`transl:nNTF` 17, 441, 2521
`_zrefclever_get_ref:n`
..... 57, 58, 1688, 1704,
1716, 1721, 1744, 1758, 1762, 1774,
1778, 1813, 1833, 2003, 2016, 2023,
2047, 2059, 2063, 2073, 2077, 2093
`_zrefclever_get_ref_first:` ...
..... 57, 58, 62, 1826, 1884, 2143
`_zrefclever_get_ref_font:nN` . 8,
15, 28, 66, 67, 1649, 1651, 1653, 2532
`_zrefclever_get_ref_string:nN` .
.... 8, 9, 15, 28, 66, 1090, 1518,
1520, 1522, 1631, 1633, 1635, 1637,
1639, 1641, 1643, 1645, 1647, 2475
`_zrefclever_get_type_transl:nnnN`
..... 9, 410, 424
`_zrefclever_get_type_transl:nnnNTF`
..... 16, 409, 2341, 2370, 2376, 2510
`\l_zrefclever_label_a_tl`
. 44, 1473, 1530, 1550, 1566, 1616,
1617, 1623, 1675, 1688, 1704, 1721,
1762, 1778, 1806, 1813, 1941, 1945,
1955, 1980, 2003, 2024, 2063, 2077
`\l_zrefclever_label_b_tl`
..... 44, 1473,
1533, 1538, 1555, 1568, 1573, 1945
`\l_zrefclever_label_count_int` ..
..... 44, 1471,
1512, 1629, 1668, 1927, 1950, 2087
`\l_zrefclever_label_enclcnt_a_`
`tl` 1112, 1237, 1239, 1240,
1261, 1288, 1329, 1369, 1377, 1378
`\l_zrefclever_label_enclcnt_b_`
`tl` 1112, 1241, 1243, 1244,
1262, 1292, 1325, 1370, 1379, 1380
`\l_zrefclever_label_enclval_a_`
`tl` 1112, 1245, 1247,
1248, 1340, 1373, 1381, 1382, 1389
`\l_zrefclever_label_enclval_b_`
`tl` 1112, 1249, 1251,
1252, 1304, 1375, 1383, 1384, 1391
`\l_zrefclever_label_type_a_tl` ..
..... 66, 1112, 1166, 1169,
1172, 1177, 1186, 1195, 1203, 1208,
1421, 1449, 1543, 1547, 1580, 1588,
1594, 1620, 1677, 1957, 2483, 2488,
2495, 2504, 2512, 2539, 2544, 2551
`\l_zrefclever_label_type_b_tl` ..
..... 1112,
1179, 1187, 1196, 1204, 1209, 1424,
1452, 1544, 1552, 1581, 1590, 1595
`_zrefclever_label_type_put_`
`new_right:n` 36, 37, 1128, 1164
`\l_zrefclever_label_types_seq` ..
.... 37, 1121, 1124, 1168, 1171, 1447
`_zrefclever_labels_in_sequence:nn`
..... 45, 65, 1804, 1944, 2422
`\g_zrefclever_languages_prop` ...
..... 11, 235, 240, 242, 250,
253, 254, 265, 412, 428, 707, 752, 970
`\l_zrefclever_last_of_type_bool`
..... 44, 1468, 1564, 1569, 1570,
1574, 1583, 1598, 1602, 1608, 1658
`\l_zrefclever_lastsep_tl` . 1488,
1640, 1703, 1720, 1743, 1761, 1773
`\l_zrefclever_link_star_bool` ...
..... 1074, 1104, 2100, 2235, 2398
`\l_zrefclever_listsep_tl`
... 1488, 1638, 1715, 1757, 2002,
2015, 2022, 2046, 2058, 2062, 2072
`\l_zrefclever_load_dict_`
`verbose_bool` ... 261, 298, 309, 318
`\g_zrefclever_loaded_dictionaries_`
`seq` 260, 269, 292, 303

`\l_zrefclever_main_language_tl` .
 22, 659,
 666, 672, 676, 680, 693, 715, 738, 760
`_zrefclever_name_default:`
 2089, 2218
`\l_zrefclever_name_format_-`
`fallback_tl`
`..` 1479, 2325, 2329, 2331, 2367, 2379
`\l_zrefclever_name_format_tl` ...
`...` 1479, 2311, 2312, 2315, 2316,
 2326, 2327, 2338, 2344, 2359, 2373
`\l_zrefclever_name_in_link_bool`
 59,
 62, 1479, 1841, 2148, 2402, 2418, 2419
`\l_zrefclever_namefont_tl` 1488,
 1650, 1844, 1872, 2177, 2208, 2223
`\l_zrefclever_nameinlink_str` ...
 643, 648,
 650, 652, 654, 2400, 2406, 2408, 2412
`\l_zrefclever_namesep_tl`
`..` 1488, 1632, 2180, 2211, 2219, 2226
`\l_zrefclever_next_is_same_bool`
 44, 65, 1483,
 1938, 1966, 1982, 1988, 2442, 2468
`\l_zrefclever_next_maybe_range_-`
`bool`
`..` 44, 65, 1483, 1800, 1810, 1937,
 1962, 1972, 2434, 2441, 2460, 2467
`\l_zrefclever_noabbrev_first_-`
`bool` 583, 592, 2322
`_zrefclever_page_format_aux:` ..
 94, 98
`\g_zrefclever_page_format_tl` ...
 7, 93, 99, 102
`\l_zrefclever_pairsep_tl`
 1488, 1636, 1687, 1811
`_zrefclever_prop_put_non_-`
`empty:Nnn` 17, 450, 820, 874
`_zrefclever_provide_dict_-`
`default_transl:nn` 14, 323, 353, 370
`_zrefclever_provide_dict_type_-`
`transl:nn` 14, 323, 371, 388
`_zrefclever_provide_dictionary:n`
 9, 12–14,
 34, 262, 319, 728, 739, 747, 762, 1075
`_zrefclever_provide_dictionary_-`
`verbose:n` ... 13, 315, 694, 702, 717
`\l_zrefclever_range_beg_label_-`
`tl` 44, 1483, 1511,
 1716, 1739, 1745, 1755, 1759, 1771,
 1775, 1926, 1964, 1979, 2013, 2017,
 2044, 2048, 2056, 2060, 2070, 2074
`\l_zrefclever_range_count_int` ..
 44,

1483, 1514, 1696, 1730, 1929, 1965,
 1976, 1981, 1987, 1995, 2036, 2082
`\l_zrefclever_range_same_count_-`
`int` 44,
 1483, 1515, 1683, 1718, 1731, 1930,
 1967, 1983, 1989, 2020, 2037, 2083
`\l_zrefclever_rangesep_tl`
 1488, 1634, 1777, 1812, 2076
`_zrefclever_ref_default:`
 2089, 2140, 2146, 2212, 2291
`\l_zrefclever_ref_language_tl` ..
 22, 23, 658, 679,
 692, 695, 700, 703, 709, 714, 718,
 728, 737, 740, 745, 748, 754, 759,
 763, 1075, 2342, 2371, 2377, 2511, 2517
`\c_zrefclever_ref_options_font_-`
`seq` 9, 15, 181
`\c_zrefclever_ref_options_-`
`necessarily_not_type_specific_-`
`seq` 15, 181, 345, 928, 1002
`\c_zrefclever_ref_options_-`
`necessarily_type_specific_seq`
 181, 376, 1046
`\c_zrefclever_ref_options_-`
`possibly_type_specific_seq` ..
 15, 181, 362, 1023
`\l_zrefclever_ref_options_prop` .
`...` 28, 30, 894, 904, 905, 2478, 2535
`\c_zrefclever_ref_options_-`
`reference_seq` 181, 896
`\c_zrefclever_ref_options_-`
`typesetup_seq` 181, 940
`\l_zrefclever_ref_property_tl` ..
 18,
 456, 461, 463, 469, 472, 488, 497,
 1125, 1157, 1541, 2095, 2119, 2133,
 2152, 2189, 2230, 2268, 2284, 2424
`\l_zrefclever_ref_typeset_font_-`
`tl` 770, 772, 1085
`\l_zrefclever_reffont_in_tl` 1488,
 1654, 2107, 2131, 2185, 2242, 2280
`\l_zrefclever_reffont_out_tl` ...
 1488, 1652,
 2104, 2128, 2182, 2202, 2239, 2277
`\l_zrefclever_refpos_in_tl` 1488,
 1648, 2120, 2134, 2190, 2269, 2285
`\l_zrefclever_refpos_out_tl` 1488,
 1644, 2123, 2136, 2203, 2272, 2287
`\l_zrefclever_refpre_in_tl` 1488,
 1646, 2117, 2132, 2186, 2265, 2281
`\l_zrefclever_refpre_out_tl` 1488,
 1642, 2105, 2129, 2183, 2240, 2278
`\l_zrefclever_setup_type_tl` ...
 14, 179, 289, 327, 340, 341,

352, 369, 383, 924, 952, 960, 973,
 997, 998, 1009, 1030, 1039, 1053, 1061
 \l_zrefclever_sort_decided_bool
 1118,
 1254, 1255, 1269, 1280, 1309, 1313,
 1317, 1351, 1355, 1359, 1387, 1399
 _zrefclever_sort_default:nn ...
 37, 1159, 1175
 _zrefclever_sort_default_-
 different_types:nn
 19, 36, 42, 1213, 1407
 _zrefclever_sort_default_same_-
 type:nn 35, 39, 1211, 1235
 _zrefclever_sort_labels:
 36, 37, 43, 1083, 1122
 _zrefclever_sort_page:nn
 43, 1158, 1459
 \l_zrefclever_sort_prior_a_int .
 1119,
 1409, 1415, 1416, 1422, 1432, 1440
 \l_zrefclever_sort_prior_b_int .
 1119,
 1410, 1417, 1418, 1425, 1433, 1441
 \l_zrefclever_tlastsep_tl
 1488, 1523, 1915
 \l_zrefclever_tlistsep_tl
 1488, 1521, 1893
 \l_zrefclever_tpairsep_tl
 1488, 1519, 1909
 \l_zrefclever_type_<type>-
 options_prop 30
 \l_zrefclever_type_count_int ...
 44, 62, 1471, 1513, 1890,
 1892, 1901, 1928, 2309, 2321, 2415
 \l_zrefclever_type_first_label_-
 tl 44, 59, 1473, 1509, 1674, 1792,
 1801, 1805, 1833, 1849, 1853, 1858,
 1864, 1924, 1954, 2145, 2151, 2158,
 2161, 2166, 2172, 2188, 2229, 2247,
 2250, 2255, 2261, 2267, 2283, 2297
 \l_zrefclever_type_first_label_-
 type_tl 44, 62, 1473, 1510, 1676,
 1796, 1925, 1956, 2300, 2336, 2343,
 2349, 2357, 2365, 2372, 2378, 2385
 _zrefclever_type_name_setup: ..
 8, 9, 59, 1821, 2295
 \l_zrefclever_type_name_tl
 59, 62,
 1479, 1867, 1873, 2178, 2209, 2216,
 2224, 2298, 2301, 2339, 2345, 2347,
 2360, 2368, 2374, 2380, 2382, 2399
 \l_zrefclever_typeset_compress_-
 bool 552, 555, 1939
 \l_zrefclever_typeset_labels_-
 seq 44, 1468, 1505, 1529, 1531, 1537
 \l_zrefclever_typeset_last_bool
 44, 1468,
 1526, 1527, 1534, 1563, 1898, 2414
 \l_zrefclever_typeset_name_bool
 503, 510, 515, 520, 1823, 1837
 \l_zrefclever_typeset_queue_-
 curr_tl 44,
 57, 62, 1473, 1508, 1685, 1701,
 1710, 1741, 1752, 1768, 1790,
 1808, 1825, 1832, 1839, 1883, 1905,
 1910, 1916, 1922, 1923, 2000, 2011,
 2042, 2054, 2068, 2314, 2409, 2413
 \l_zrefclever_typeset_queue_-
 prev_tl . 44, 1473, 1507, 1894, 1921
 \l_zrefclever_typeset_range_-
 bool 561, 564, 1082, 1788
 \l_zrefclever_typeset_ref_bool .
 502, 509, 514, 519, 1823, 1830
 _zrefclever_typeset_refs:
 44-46, 1086, 1503
 _zrefclever_typeset_refs_last_-
 of_type: . 49, 57, 59, 62, 1660, 1665
 _zrefclever_typeset_refs_not_-
 last_of_type:
 45, 49, 57, 65, 1662, 1933
 \l_zrefclever_typeset_sort_bool
 528, 531, 1081
 \l_zrefclever_typesort_seq
 19, 42, 537, 542, 543, 549, 1411
 \l_zrefclever_use_hyperref_bool
 602, 609,
 614, 619, 629, 635, 2099, 2234, 2397
 \l_zrefclever_warn_hyperref_-
 bool 603, 610, 615, 620, 633
 _zrefclever_zcref:nnn .. 1068, 1069
 _zrefclever_zcref:nnnn 34, 36, 1069
 \l_zrefclever_zcref_labels_seq .
 36,
 37, 1073, 1100, 1104, 1127, 1130, 1506
 \l_zrefclever_zcref_note_tl ...
 773, 776, 1088, 1092
 \l_zrefclever_zcref_with_check_-
 bool 780, 795, 1078, 1096
 \l_zrefclever_zrefcheck_-
 available_bool
 779, 790, 801, 1077, 1095