The **zref-clever** package implementation[*]

Gustavo Barros[†]

2021-09-29

Contents

---

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

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

1   Initial setup

Start the DocStrip guards.

```
1  ⟨*package⟩
```

Identify the internal prefix (LaTeX3 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 (ltcmdhooks), which is bound to be useful for our purposes, and was released with the 2021-06-01 kernel.

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

Identify the package.

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

2   Dependencies

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

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

3 zref setup

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

> __zrefclever_get_enclosing_counters:n {⟨*counter*⟩}
> __zrefclever_get_enclosing_counters_value:n {⟨*counter*⟩}

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

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

```
55 \cs_generate_variant:Nn \__zrefclever_get_enclosing_counters:n { V , e }
56 \cs_generate_variant:Nn \__zrefclever_get_enclosing_counters_value:n { V , e }
```

(*End definition for* __zrefclever_get_enclosing_counters:n *and* __zrefclever_get_enclosing_-counters_value:n.)

__zrefclever_counter_reset_by:n   Auxiliary function for __zrefclever_get_enclosing_counters:n and __zrefclever_-get_enclosing_counters_value:n. They are broken in parts to be able to use the expandable mapping functions. __zrefclever_counter_reset_by:n leaves in the stream the "enclosing counter" which resets ⟨*counter*⟩.

> __zrefclever_counter_reset_by:n {⟨*counter*⟩}

```
57 \cs_new:Npn \__zrefclever_counter_reset_by:n #1
58   {
59     \bool_if:nTF
60       { \prop_if_in_p:Nn \l__zrefclever_counter_resetby_prop {#1} }
61       { \prop_item:Nn  \l__zrefclever_counter_resetby_prop {#1} }
62       {
63         \seq_map_tokens:Nn \l__zrefclever_counter_resetters_seq
64           { \__zrefclever_counter_reset_by_aux:nn {#1} }
65       }
66   }
67 \cs_new:Npn \__zrefclever_counter_reset_by_aux:nn #1#2
68   {
```

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

(*End definition for* `\__zrefclever_counter_reset_by:n.`)

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

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

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

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

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

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

4 Plumbing

4.1 Messages

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

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

4.2 Reference format

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

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

/ enforced in __zrefclever_get_ref_string:nN, __zrefclever_get_ref_font:nN, and __zrefclever_type_name_setup: which are the basic functions to retrieve proper values for reference format settings.

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

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

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

\l__zrefclever_setup_type_tl
\l_zrefclever_dict_language_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.

```
175 \tl_new:N \l__zrefclever_setup_type_tl
176 \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
ever_ref_options_possibly_type_specific_seq
r_ref_options_necessarily_type_specific_seq
\c__zrefclever_ref_options_font_seq
\c__zrefclever_ref_options_typesetup_seq
\c__zrefclever_ref_options_reference_seq

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.

```
177 \seq_const_from_clist:Nn
178   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
179   {
180     tpairsep ,
181     tlistsep ,
182     tlastsep ,
183     notesep ,
184   }
185 \seq_const_from_clist:Nn
186   \c__zrefclever_ref_options_possibly_type_specific_seq
187   {
188     namesep ,
189     pairsep ,
190     listsep ,
191     lastsep ,
192     rangesep ,
193     refpre ,
194     refpos ,
195     refpre-in ,
196     refpos-in ,
197   }
```

9

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:`.

```
198 \seq_const_from_clist:Nn
199   \c__zrefclever_ref_options_necessarily_type_specific_seq
200   {
201     Name-sg ,
202     name-sg ,
203     Name-pl ,
204     name-pl ,
205     Name-sg-ab ,
206     name-sg-ab ,
207     Name-pl-ab ,
208     name-pl-ab ,
209   }
```

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

```
210 \seq_const_from_clist:Nn
211   \c__zrefclever_ref_options_font_seq
212   {
213     namefont ,
214     reffont ,
215     reffont-in ,
216   }
217 \seq_new:N \c__zrefclever_ref_options_typesetup_seq
218 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
219   \c__zrefclever_ref_options_possibly_type_specific_seq
220   \c__zrefclever_ref_options_necessarily_type_specific_seq
221 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
222   \c__zrefclever_ref_options_typesetup_seq
223   \c__zrefclever_ref_options_font_seq
224 \seq_new:N \c__zrefclever_ref_options_reference_seq
225 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
226   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
227   \c__zrefclever_ref_options_possibly_type_specific_seq
228 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
229   \c__zrefclever_ref_options_reference_seq
230   \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 of 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".

```
231 \prop_new:N \g__zrefclever_languages_prop
```

(*End definition for* `\g__zrefclever_languages_prop`.)

\zcDeclareLanguage Declare a new language for use with zref-clever. ⟨*language*⟩ is taken to be both the "language name" and the "dictionary name". If ⟨*language*⟩ is already known, just warn. \zcDeclareLanguage is preamble only.

> \zcDeclareLanguage {⟨*language*⟩}

```
232 \NewDocumentCommand \zcDeclareLanguage { m }
233   {
234     \tl_if_empty:nF {#1}
235       {
236         \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
237           { \msg_warning:nnn { zref-clever } { language-declared } {#1} }
238           { \prop_gput:Nnn \g__zrefclever_languages_prop {#1} {#1} }
239       }
240   }
241 \@onlypreamble \zcDeclareLanguage
```

(*End definition for* \zcDeclareLanguage.)

\zcDeclareLanguageAlias Declare ⟨*language alias*⟩ to be an alias of ⟨*aliased language*⟩. ⟨*aliased language*⟩ must be already known to zref-clever, as stored in \g__zrefclever_languages_prop. \zcDeclareLanguageAlias is preamble only.

> \zcDeclareLanguageAlias {⟨*language alias*⟩} {⟨*aliased language*⟩}

```
242 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
243   {
244     \tl_if_empty:nF {#1}
245       {
246         \prop_if_in:NnTF \g__zrefclever_languages_prop {#2}
247           {
248             \exp_args:NNnx
249               \prop_gput:Nnn \g__zrefclever_languages_prop {#1}
250                 { \prop_item:Nn \g__zrefclever_languages_prop {#2} }
251           }
252           { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
253       }
254   }
255 \@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 .lbx files. I'm not really well acquainted with this machinery, but as far as I grasp, they all rely on some variation of \ProvidesFile and \input. And they can be safely \input without generating spurious content, because they rely on being loaded before the document has actually started. As far as I can tell, babel's "on the fly" functionality is not based on the .ldf files, but on the .ini files, and on \babelprovide. And the .ini files are not in this form, but actually resemble "configuration files" of sorts, which means they are read and processed somehow else than with just \input. So we do the more or less the same here. It seems a reasonable way to ensure we can load dictionaries on the fly robustly mid-document, without getting paranoid with the last bit of white-space in them, and without introducing any undue content on the stream when we cannot afford to do it. Hence, zref-clever's built-in dictionary files are a set of *key-value options* which are read from the file, and fed to \keys_set:nn{zref-clever/dictionary} by __zrefclever_provide_dictionary:n. And they use the same syntax and options as \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.

```
256 \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.

```
257 \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⟩}

```
258  \cs_new_protected:Npn \__zrefclever_provide_dictionary:n #1
259    {
260      \group_begin:
261      \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
262        \l__zrefclever_dict_language_tl
263        {
264          \seq_if_in:NVF
265            \g__zrefclever_loaded_dictionaries_seq
266            \l__zrefclever_dict_language_tl
267            {
268              \exp_args:Nx \file_get:nnNTF
269                { zref-clever- \l__zrefclever_dict_language_tl .dict }
270                { \ExplSyntaxOn }
271                \l_tmpa_tl
272                {
273                  \prop_if_exist:cF
274                    {
275                      g__zrefclever_dict_
276                      \l__zrefclever_dict_language_tl _prop
277                    }
278                    {
279                      \prop_new:c
280                        {
281                          g__zrefclever_dict_
282                          \l__zrefclever_dict_language_tl _prop
283                        }
284                    }
285                  \tl_clear:N \l__zrefclever_setup_type_tl
286                  \exp_args:NnV
287                    \keys_set:nn { zref-clever / dictionary } \l_tmpa_tl
288                  \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
289                    \l__zrefclever_dict_language_tl
290                  \msg_note:nnx { zref-clever } { dict-loaded }
291                    { \l__zrefclever_dict_language_tl }
292                }
293                {
294                  \bool_if:NT \l__zrefclever_load_dict_verbose_bool
295                    {
296                      \msg_warning:nnx { zref-clever } { dict-not-available }
297                        { \l__zrefclever_dict_language_tl }
298                    }
```

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.

```
299                  \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
300                    \l__zrefclever_dict_language_tl
301                }
302            }
303        }
```

```
304        {
305          \bool_if:NT \l__zrefclever_load_dict_verbose_bool
306            { \msg_warning:nnn { zref-clever } { unknown-language-load } {#1} }
307        }
308      \group_end:
309    }
310  \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⟩}`

```
311  \cs_new_protected:Npn \__zrefclever_provide_dictionary_verbose:n #1
312    {
313      \group_begin:
314      \bool_set_true:N \l__zrefclever_load_dict_verbose_bool
315      \__zrefclever_provide_dictionary:n {#1}
316      \group_end:
317    }
318  \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 ⟨*key*⟩ and ⟨*translation*⟩ as arguments, but `\__zrefclever_provide_dict_-`
`type_transl:nn` relies on the current value of `\l__zrefclever_setup_type_tl`, as set by the type key.

> `\__zrefclever_provide_dict_type_transl:nn {⟨key⟩} {⟨translation⟩}`
> `\__zrefclever_provide_dict_default_transl:nn {⟨key⟩} {⟨translation⟩}`

```
319  \cs_new_protected:Npn \__zrefclever_provide_dict_type_transl:nn #1#2
320    {
321      \exp_args:Nnx \prop_gput_if_new:cnn
322        { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
323        { type- \l__zrefclever_setup_type_tl - #1 } {#2}
324    }
325  \cs_new_protected:Npn \__zrefclever_provide_dict_default_transl:nn #1#2
326    {
327      \prop_gput_if_new:cnn
328        { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
329        { default- #1 } {#2}
330    }
```

(*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.

```
331 \keys_define:nn { zref-clever / dictionary }
332   {
333     type .code:n =
334       {
335         \tl_if_empty:nTF {#1}
336           { \tl_clear:N \l__zrefclever_setup_type_tl }
337           { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
338       } ,
339   }
340 \seq_map_inline:Nn
341   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
342   {
343     \keys_define:nn { zref-clever / dictionary }
344       {
345         #1 .value_required:n = true ,
346         #1 .code:n =
347           {
348             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
349               { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
350               {
351                 \msg_info:nnn { zref-clever }
352                   { option-not-type-specific } {#1}
353               }
354           } ,
355       }
356   }
357 \seq_map_inline:Nn
358   \c__zrefclever_ref_options_possibly_type_specific_seq
359   {
360     \keys_define:nn { zref-clever / dictionary }
361       {
362         #1 .value_required:n = true ,
363         #1 .code:n =
364           {
365             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
366               { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
367               { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
368           } ,
369       }
370   }
371 \seq_map_inline:Nn
372   \c__zrefclever_ref_options_necessarily_type_specific_seq
373   {
374     \keys_define:nn { zref-clever / dictionary }
375       {
376         #1 .value_required:n = true ,
377         #1 .code:n =
378           {
379             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
380               {
381                 \msg_info:nnn { zref-clever }
382                   { option-only-type-specific } {#1}
383               }
384               { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
```

15

```
385                } ,
386             }
387       }
```

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.

```
388  \prop_new:N \g__zrefclever_fallback_dict_prop
389  \prop_gset_from_keyval:Nn \g__zrefclever_fallback_dict_prop
390    {
391      tpairsep  = {,~} ,
392      tlistsep  = {,~} ,
393      tlastsep  = {,~} ,
394      notesep   = {~} ,
395      namesep   = {\nobreakspace} ,
396      pairsep   = {,~} ,
397      listsep   = {,~} ,
398      lastsep   = {,~} ,
399      rangesep  = {\textendash} ,
400      refpre    = {} ,
401      refpos    = {} ,
402      refpre-in = {} ,
403      refpos-in = {} ,
404    }
```

Get translations

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

> `\__zrefclever_get_type_transl:nnnNF {⟨language⟩} {⟨type⟩} {⟨key⟩}`
> `⟨tl variable⟩ {⟨false code⟩}`

```
405  \prg_new_protected_conditional:Npnn
406    \__zrefclever_get_type_transl:nnnN #1#2#3#4 { F }
407    {
408      \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
409        \l__zrefclever_dict_language_tl
410        {
411          \prop_get:cnNTF
412            { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
```

16

```
413          { type- #2 - #3 } #4
414          { \prg_return_true:  }
415          { \prg_return_false: }
416      }
417     { \prg_return_false: }
418   }
419 \prg_generate_conditional_variant:Nnn
420   \__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 ⟨*key*⟩ for ⟨*language*⟩, and store it in ⟨*tl variable*⟩ if found. If not found, leave the ⟨*false code*⟩ on the stream, in which case the value of ⟨*tl variable*⟩ should not be relied upon.

> __zrefclever_get_default_transl:nnNF {⟨*language*⟩} {⟨*key*⟩}
>   ⟨*tl variable*⟩ {⟨*false code*⟩}

```
421 \prg_new_protected_conditional:Npnn
422   \__zrefclever_get_default_transl:nnN #1#2#3 { F }
423   {
424     \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
425       \l__zrefclever_dict_language_tl
426       {
427         \prop_get:cnNTF
428           { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
429           { default- #2 } #3
430           { \prg_return_true:  }
431           { \prg_return_false: }
432       }
433     { \prg_return_false: }
434   }
435 \prg_generate_conditional_variant:Nnn
436   \__zrefclever_get_default_transl:nnN { xnN } { F }
```

(*End definition for* __zrefclever_get_default_transl:nnNF.)

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

> __zrefclever_get_fallback_transl:nNF {⟨*key*⟩}
>   ⟨*tl variable*⟩ {⟨*false code*⟩}

```
437 % {<key>}<tl var to set>
438 \prg_new_protected_conditional:Npnn
439   \__zrefclever_get_fallback_transl:nN #1#2 { F }
440   {
441     \prop_get:NnNTF \g__zrefclever_fallback_dict_prop
442       { #1 } #2
443       { \prg_return_true:  }
444       { \prg_return_false: }
445   }
```

(*End definition for* __zrefclever_get_fallback_transl:nNF.)

4.5 Options

Auxiliary

__zrefclever_prop_put_non_empty:Nnn   If ⟨*value*⟩ is empty, remove ⟨*key*⟩ from ⟨*property list*⟩. Otherwise, add ⟨*key*⟩ = ⟨*value*⟩ to ⟨*property list*⟩.

> `\__zrefclever_prop_put_non_empty:Nnn` ⟨*property list*⟩ {⟨*key*⟩} {⟨*value*⟩}

```
446 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
447   {
448     \tl_if_empty:nTF {#3}
449       { \prop_remove:Nn #1 {#2} }
450       { \prop_put:Nnn #1 {#2} {#3} }
451   }
```

(*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.

```
452 \tl_new:N \l__zrefclever_ref_property_tl
453 \keys_define:nn { zref-clever / reference }
454   {
455     ref .choice: ,
456     ref / zc@thecnt .code:n =
457       { \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt } } ,
458     ref / page .code:n =
459       { \tl_set:Nn \l__zrefclever_ref_property_tl { page } } ,
460     ref / title .code:n =
461       {
462         \AddToHook { begindocument }
463           {
464             \@ifpackageloaded { zref-titleref }
465               { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
466               {
467                 \msg_warning:nn { zref-clever } { missing-zref-titleref }
468                 \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
469               }
470           }
471       } ,
472     ref .initial:n = zc@thecnt ,
473     ref .default:n = zc@thecnt ,
474     page .meta:n = { ref = page },
```

18

```
475    page .value_forbidden:n = true ,
476  }
477 \AddToHook { begindocument }
478  {
479    \@ifpackageloaded { zref-titleref }
480      {
481        \keys_define:nn { zref-clever / reference }
482          {
483            ref / title .code:n =
484              { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
485          }
486      }
487      {
488        \keys_define:nn { zref-clever / reference }
489          {
490            ref / title .code:n =
491              {
492                \msg_warning:nn { zref-clever } { missing-zref-titleref }
493                \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
494              }
495          }
496      }
497  }
```

typeset option

```
498 \bool_new:N \l__zrefclever_typeset_ref_bool
499 \bool_new:N \l__zrefclever_typeset_name_bool
500 \keys_define:nn { zref-clever / reference }
501  {
502    typeset .choice: ,
503    typeset / both .code:n =
504      {
505        \bool_set_true:N \l__zrefclever_typeset_ref_bool
506        \bool_set_true:N \l__zrefclever_typeset_name_bool
507      } ,
508    typeset / ref .code:n =
509      {
510        \bool_set_true:N \l__zrefclever_typeset_ref_bool
511        \bool_set_false:N \l__zrefclever_typeset_name_bool
512      } ,
513    typeset / name .code:n =
514      {
515        \bool_set_false:N \l__zrefclever_typeset_ref_bool
516        \bool_set_true:N \l__zrefclever_typeset_name_bool
517      } ,
518    typeset .initial:n = both ,
519    typeset .value_required:n = true ,
520
521    noname .meta:n = { typeset = ref },
522    noname .value_forbidden:n = true ,
523  }
```

sort option

```
524 \bool_new:N \l__zrefclever_typeset_sort_bool
525 \keys_define:nn { zref-clever / reference }
526   {
527     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
528     sort .initial:n = true ,
529     sort .default:n = true ,
530     nosort .meta:n = { sort = false },
531     nosort .value_forbidden:n = true ,
532   }
```

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 $\seq_map_indexed_inline:Nn$.

```
533 \seq_new:N \l__zrefclever_typesort_seq
534 \keys_define:nn { zref-clever / reference }
535   {
536     typesort .code:n =
537       {
538         \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
539         \seq_reverse:N \l__zrefclever_typesort_seq
540       } ,
541     typesort .initial:n =
542       { part , chapter , section , paragraph },
543     typesort .value_required:n = true ,
544     notypesort .code:n =
545       { \seq_clear:N \l__zrefclever_typesort_seq } ,
546     notypesort .value_forbidden:n = true ,
547   }
```

comp option

```
548 \bool_new:N \l__zrefclever_typeset_compress_bool
549 \keys_define:nn { zref-clever / reference }
550   {
551     comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
552     comp .initial:n = true ,
553     comp .default:n = true ,
554     nocomp .meta:n = { comp = false },
555     nocomp .value_forbidden:n = true ,
556   }
```

range option

```
557 \bool_new:N \l__zrefclever_typeset_range_bool
558 \keys_define:nn { zref-clever / reference }
559   {
560     range .bool_set:N = \l__zrefclever_typeset_range_bool ,
561     range .initial:n = false ,
562     range .default:n = true ,
563   }
```

cap and capfirst options

```
564 \bool_new:N \l__zrefclever_capitalize_bool
565 \bool_new:N \l__zrefclever_capitalize_first_bool
566 \keys_define:nn { zref-clever / reference }
567   {
568     cap .bool_set:N = \l__zrefclever_capitalize_bool ,
569     cap .initial:n = false ,
570     cap .default:n = true ,
571     nocap .meta:n = { cap = false },
572     nocap .value_forbidden:n = true ,
573
574     capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,
575     capfirst .initial:n = false ,
576     capfirst .default:n = true ,
577   }
```

abbrev and noabbrevfirst options

```
578 \bool_new:N \l__zrefclever_abbrev_bool
579 \bool_new:N \l__zrefclever_noabbrev_first_bool
580 \keys_define:nn { zref-clever / reference }
581   {
582     abbrev .bool_set:N = \l__zrefclever_abbrev_bool ,
583     abbrev .initial:n = false ,
584     abbrev .default:n = true ,
585     noabbrev .meta:n = { abbrev = false },
586     noabbrev .value_forbidden:n = true ,
587
588     noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
589     noabbrevfirst .initial:n = false ,
590     noabbrevfirst .default:n = true ,
591   }
```

S option

```
592 \keys_define:nn { zref-clever / reference }
593   {
594     S .meta:n =
595       { capfirst = true , noabbrevfirst = true },
596     S .value_forbidden:n = true ,
597   }
```

hyperref option

```
598 \bool_new:N \l__zrefclever_use_hyperref_bool
599 \bool_new:N \l__zrefclever_warn_hyperref_bool
600 \keys_define:nn { zref-clever / reference }
601   {
602     hyperref .choice: ,
603     hyperref / auto .code:n =
604       {
605         \bool_set_true:N \l__zrefclever_use_hyperref_bool
606         \bool_set_false:N \l__zrefclever_warn_hyperref_bool
607       } ,
608     hyperref / true .code:n =
609       {
610         \bool_set_true:N \l__zrefclever_use_hyperref_bool
611         \bool_set_true:N \l__zrefclever_warn_hyperref_bool
```

```
612       } ,
613     hyperref / false .code:n =
614       {
615         \bool_set_false:N \l__zrefclever_use_hyperref_bool
616         \bool_set_false:N \l__zrefclever_warn_hyperref_bool
617       } ,
618     hyperref .initial:n = auto ,
619     hyperref .default:n = auto
620   }
621 \AddToHook { begindocument }
622   {
623     \@ifpackageloaded { hyperref }
624       {
625         \bool_if:NT \l__zrefclever_use_hyperref_bool
626           { \RequirePackage { zref-hyperref } }
627       }
628       {
629         \bool_if:NT \l__zrefclever_warn_hyperref_bool
630           { \msg_warning:nn { zref-clever } { missing-hyperref } }
631         \bool_set_false:N \l__zrefclever_use_hyperref_bool
632       }
633     \keys_define:nn { zref-clever / reference }
634       {
635         hyperref .code:n =
636           { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }
637       }
638   }
```

nameinlink option

```
639 \str_new:N \l__zrefclever_nameinlink_str
640 \keys_define:nn { zref-clever / reference }
641   {
642     nameinlink .choice: ,
643     nameinlink / true .code:n =
644       { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
645     nameinlink / false .code:n =
646       { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
647     nameinlink / single .code:n =
648       { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
649     nameinlink / tsingle .code:n =
650       { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
651     nameinlink .initial:n = tsingle ,
652     nameinlink .default:n = true ,
653   }
```

lang option

\l__zrefclever_current_language_tl is an internal alias for babel's \languagename
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`.

```
654  \tl_new:N \l__zrefclever_ref_language_tl
655  \tl_new:N \l__zrefclever_main_language_tl
656  \tl_new:N \l__zrefclever_current_language_tl
657  \AddToHook { begindocument }
658    {
659      \@ifpackageloaded { babel }
660        {
661          \tl_set:Nn \l__zrefclever_current_language_tl { \languagename }
662          \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
663        }
664        {
665          \@ifpackageloaded { polyglossia }
666            {
667              \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
668              \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
669            }
670            {
671              \tl_set:Nn \l__zrefclever_current_language_tl { english }
672              \tl_set:Nn \l__zrefclever_main_language_tl { english }
673            }
674        }
```

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.

```
675      \tl_set:Nn \l__zrefclever_ref_language_tl
676        { \l__zrefclever_main_language_tl }
677    }
678  \keys_define:nn { zref-clever / reference }
679    {
680      lang .code:n =
681        {
```

```
682        \AddToHook { begindocument }
683          {
684            \str_case:nnF {#1}
685              {
686                { main }
687                {
688                  \tl_set:Nn \l__zrefclever_ref_language_tl
689                    { \l__zrefclever_main_language_tl }
690                  \__zrefclever_provide_dictionary_verbose:x
691                    { \l__zrefclever_ref_language_tl }
692                }
693
694                { current }
695                {
696                  \tl_set:Nn \l__zrefclever_ref_language_tl
697                    { \l__zrefclever_current_language_tl }
698                  \__zrefclever_provide_dictionary_verbose:x
699                    { \l__zrefclever_ref_language_tl }
700                }
701              }
702              {
703                \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
704                  {
705                    \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
706                  }
707                  {
708                    \msg_warning:nnn { zref-clever }
709                      { unknown-language-opt } {#1}
710                    \tl_set:Nn \l__zrefclever_ref_language_tl
711                      { \l__zrefclever_main_language_tl }
712                  }
713                \__zrefclever_provide_dictionary_verbose:x
714                  { \l__zrefclever_ref_language_tl }
715              }
716          }
717        } ,
718      lang .value_required:n = true ,
719    }

720  \AddToHook { begindocument / before }
721    {
722      \AddToHook { begindocument }
723        {
```

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.

```
724          \__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.

```
725          \keys_define:nn { zref-clever / reference }
726            {
727              lang .code:n =
```

24

```
728                    {
729                      \str_case:nnF {#1}
730                        {
731                          { main }
732                          {
733                            \tl_set:Nn \l__zrefclever_ref_language_tl
734                              { \l__zrefclever_main_language_tl }
735                            \__zrefclever_provide_dictionary:x
736                              { \l__zrefclever_ref_language_tl }
737                          }
738
739                          { current }
740                          {
741                            \tl_set:Nn \l__zrefclever_ref_language_tl
742                              { \l__zrefclever_current_language_tl }
743                            \__zrefclever_provide_dictionary:x
744                              { \l__zrefclever_ref_language_tl }
745                          }
746                        }
747                        {
748                          \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
749                            {
750                              \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
751                            }
752                            {
753                              \msg_warning:nnn { zref-clever }
754                                { unknown-language-opt } {#1}
755                              \tl_set:Nn \l__zrefclever_ref_language_tl
756                                { \l__zrefclever_main_language_tl }
757                            }
758                          \__zrefclever_provide_dictionary:x
759                            { \l__zrefclever_ref_language_tl }
760                        }
761                    } ,
762                  lang .value_required:n = true ,
763                }
764          }
765    }
```

font option

`font` *can't be used as a package option*, since the options get expanded by LaTeX 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`.

```
766 \tl_new:N \l__zrefclever_ref_typeset_font_tl
767 \keys_define:nn { zref-clever / reference }
768   { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }
```

note option

```
769 \tl_new:N \l__zrefclever_zcref_note_tl
770 \keys_define:nn { zref-clever / reference }
771   {
```

```
772    note .tl_set:N = \l__zrefclever_zcref_note_tl ,
773    note .value_required:n = true ,
774  }
```

check option

Integration with zref-check.

```
775 \bool_new:N \l__zrefclever_zrefcheck_available_bool
776 \bool_new:N \l__zrefclever_zcref_with_check_bool
777 \keys_define:nn { zref-clever / reference }
778   {
779     check .code:n =
780       { \msg_warning:nn { zref-clever } { check-document-only } } ,
781   }
782 \AddToHook { begindocument }
783   {
784     \@ifpackageloaded { zref-check }
785       {
786         \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
787         \keys_define:nn { zref-clever / reference }
788           {
789             check .code:n =
790               {
791                 \bool_set_true:N \l__zrefclever_zcref_with_check_bool
792                 \keys_set:nn { zref-check / zcheck } {#1}
793               }
794           }
795       }
796       {
797         \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
798         \keys_define:nn { zref-clever / reference }
799           {
800             check .code:n =
801               { \msg_warning:nn { zref-clever } { missing-zref-check } }
802           }
803       }
804   }
```

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`.

```
805 \prop_new:N \l__zrefclever_counter_type_prop
806 \keys_define:nn { zref-clever / label }
807   {
808     countertype .code:n =
809       {
810         \keyval_parse:nnn
811           {
812             \msg_warning:nnnn { zref-clever }
813               { key-requires-value } { countertype }
814           }
```

26

```
815            {
816              \__zrefclever_prop_put_non_empty:Nnn
817                \l__zrefclever_counter_type_prop
818            }
819          {#1}
820      } ,
821    countertype .value_required:n = true ,
822    countertype .initial:n =
823      {
824        subsection    = section ,
825        subsubsection = section ,
826        subparagraph  = paragraph ,
827        enumi         = item ,
828        enumii        = item ,
829        enumiii       = item ,
830        enumiv        = item ,
831      } ,
832  }
```

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.

```
833  \seq_new:N \l__zrefclever_counter_resetters_seq
834  \keys_define:nn { zref-clever / label }
835    {
836      counterresetters .code:n =
837        {
838          \clist_map_inline:nn {#1}
839            {
840              \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
841                {
842                  \seq_put_right:Nn
843                    \l__zrefclever_counter_resetters_seq {##1}
844                }
845            }
846        } ,
847      counterresetters .initial:n =
848        {
849          part ,
850          chapter ,
851          section ,
852          subsection ,
853          subsubsection ,
854          paragraph ,
855          subparagraph ,
856        },
```

```
857       counterresetters .value_required:n = true ,
858     }
```

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`.

```
859   \prop_new:N \l__zrefclever_counter_resetby_prop
860   \keys_define:nn { zref-clever / label }
861     {
862       counterresetby .code:n =
863         {
864           \keyval_parse:nnn
865             {
866               \msg_warning:nnn { zref-clever }
867                 { key-requires-value } { counterresetby }
868             }
869             {
870               \__zrefclever_prop_put_non_empty:Nnn
871                 \l__zrefclever_counter_resetby_prop
872             }
873             {#1}
874         } ,
875       counterresetby .value_required:n = true ,
876       counterresetby .initial:n =
877         {
```

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.

```
878           enumii  = enumi   ,
879           enumiii = enumii  ,
880           enumiv  = enumiii ,
881         } ,
882     }
```

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.

```
883 \prop_new:N \l__zrefclever_ref_options_prop
884 \seq_map_inline:Nn
885   \c__zrefclever_ref_options_reference_seq
886   {
887     \keys_define:nn { zref-clever / reference }
888       {
889         #1 .default:V = \c_novalue_tl ,
890         #1 .code:n =
891           {
892             \tl_if_novalue:nTF {##1}
893               { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} }
894               { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
895           } ,
896       }
897   }
```

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.

```
898 \keys_define:nn { }
899   {
900     zref-clever / zcsetup .inherit:n = zref-clever / label ,
901     zref-clever / zcsetup .inherit:n = zref-clever / reference ,
902   }
```

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

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

5   Configuration

5.1   \zcsetup

\zcsetup   Provide `\zcsetup`.

\zcsetup{⟨*options*⟩}

```
904 \NewDocumentCommand \zcsetup { m }
905   { \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*⟩}

```
906 \NewDocumentCommand \zcRefTypeSetup { m m }
907   {
908     \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
909       { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
910     \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
911     \keys_set:nn { zref-clever / typesetup } {#2}
912   }
```

(*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.

```
913 \seq_map_inline:Nn
914   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
915   {
916     \keys_define:nn { zref-clever / typesetup }
917       {
918         #1 .code:n =
919           {
920             \msg_warning:nnn { zref-clever }
921               { option-not-type-specific } {#1}
922           } ,
923       }
924   }
925 \seq_map_inline:Nn
926   \c__zrefclever_ref_options_typesetup_seq
927   {
928     \keys_define:nn { zref-clever / typesetup }
929       {
930         #1 .default:V = \c_novalue_tl ,
931         #1 .code:n =
932           {
933             \tl_if_novalue:nTF {##1}
934               {
935                 \prop_remove:cn
936                   {
937                     l__zrefclever_type_
```

```
938                        \l__zrefclever_setup_type_tl _options_prop
939                      }
940                      {#1}
941                  }
942                  {
943                    \prop_put:cnn
944                      {
945                        l__zrefclever_type_
946                        \l__zrefclever_setup_type_tl _options_prop
947                      }
948                      {#1} {##1}
949                  }
950            } ,
951          }
952    }
```

5.3 \zcLanguageSetup

\zcLanguageSetup is the main user interface for "language-specific" reference format-
ting, 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 follow-
ing 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⟩}

```
953  \NewDocumentCommand \zcLanguageSetup { m m }
954    {
955      \group_begin:
956      \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
957        \l__zrefclever_dict_language_tl
958        {
959          \tl_clear:N \l__zrefclever_setup_type_tl
960          \keys_set:nn { zref-clever / langsetup } {#2}
961        }
962        { \msg_warning:nnn { zref-clever } { unknown-language-transl } {#1} }
963      \group_end:
964    }
965  \@onlypreamble \zcLanguageSetup
```

(*End definition for* \zcLanguageSetup*.*)

_zrefclever_declare_type_transl:nnnn A couple of auxiliary functions for the of zref-clever/translation keys set in
_zrefclever_declare_default_transl:nnn \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⟩}

31

```
966 \cs_new_protected:Npn \__zrefclever_declare_type_transl:nnnn #1#2#3#4
967   {
968     \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
969       { type- #2 - #3 } {#4}
970   }
971 \cs_generate_variant:Nn \__zrefclever_declare_type_transl:nnnn { VVnn }
972 \cs_new_protected:Npn \__zrefclever_declare_default_transl:nnn #1#2#3
973   {
974     \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
975       { default- #2 } {#3}
976   }
977 \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.

```
978 \keys_define:nn { zref-clever / langsetup }
979   {
980     type .code:n =
981       {
982         \tl_if_empty:nTF {#1}
983           { \tl_clear:N \l__zrefclever_setup_type_tl }
984           { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
985       } ,
986   }
987 \seq_map_inline:Nn
988   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
989   {
990     \keys_define:nn { zref-clever / langsetup }
991       {
992         #1 .value_required:n = true ,
993         #1 .code:n =
994           {
995             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
996               {
997                 \__zrefclever_declare_default_transl:Vnn
998                   \l__zrefclever_dict_language_tl
999                   {#1} {##1}
1000              }
1001              {
1002                \msg_warning:nnn { zref-clever }
1003                  { option-not-type-specific } {#1}
1004              }
1005          } ,
1006      }
1007   }
1008 \seq_map_inline:Nn
1009   \c__zrefclever_ref_options_possibly_type_specific_seq
1010   {
1011     \keys_define:nn { zref-clever / langsetup }
1012       {
1013         #1 .value_required:n = true ,
1014         #1 .code:n =
```

```
1015            {
1016              \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1017                {
1018                  \__zrefclever_declare_default_transl:Vnn
1019                    \l__zrefclever_dict_language_tl
1020                    {#1} {##1}
1021                }
1022                {
1023                  \__zrefclever_declare_type_transl:VVnn
1024                    \l__zrefclever_dict_language_tl
1025                    \l__zrefclever_setup_type_tl
1026                    {#1} {##1}
1027                }
1028            } ,
1029          }
1030    }
1031  \seq_map_inline:Nn
1032    \c__zrefclever_ref_options_necessarily_type_specific_seq
1033    {
1034      \keys_define:nn { zref-clever / langsetup }
1035        {
1036          #1 .value_required:n = true ,
1037          #1 .code:n =
1038            {
1039              \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1040                {
1041                  \msg_warning:nnn { zref-clever }
1042                    { option-only-type-specific } {#1}
1043                }
1044                {
1045                  \__zrefclever_declare_type_transl:VVnn
1046                    \l__zrefclever_dict_language_tl
1047                    \l__zrefclever_setup_type_tl
1048                    {#1} {##1}
1049                }
1050            } ,
1051        }
1052    }
```

6   User interface

6.1   \zcref

\zcref   The main user command of the package.

> \zcref⟨*⟩[⟨*options*⟩]{⟨*labels*⟩}

```
1053  \NewDocumentCommand \zcref { s O { } m }
1054    { \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⟩}

1055 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
1056   {
1057      \group_begin:
```
Set options.
```
1058         \keys_set:nn { zref-clever / reference } {#3}
```
Store arguments values.
```
1059         \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
1060         \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 lan-
guage may have changed outside our control. __zrefclever_provide_dictionary:x
does nothing if the dictionary is already loaded.
```
1061         \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }
```
Integration with zref-check.
```
1062         \bool_lazy_and:nnT
1063           { \l__zrefclever_zrefcheck_available_bool }
1064           { \l__zrefclever_zcref_with_check_bool }
1065           { \zrefcheck_zcref_beg_label: }
```
Sort the labels.
```
1066         \bool_lazy_or:nnT
1067           { \l__zrefclever_typeset_sort_bool }
1068           { \l__zrefclever_typeset_range_bool }
1069           { \__zrefclever_sort_labels: }
```
Typeset the references. Also, set the reference font, and group it, so that it does not leak
to the note.
```
1070         \group_begin:
1071         \l__zrefclever_ref_typeset_font_tl
1072         \__zrefclever_typeset_refs:
1073         \group_end:
```
Typeset note.
```
1074         \tl_if_empty:NF \l__zrefclever_zcref_note_tl
1075           {
1076             \__zrefclever_get_ref_string:nN { notesep } \l_tmpa_tl
1077             \l_tmpa_tl
1078             \l__zrefclever_zcref_note_tl
1079           }
```
Integration with zref-check.
```
1080         \bool_lazy_and:nnT
1081           { \l__zrefclever_zrefcheck_available_bool }
1082           { \l__zrefclever_zcref_with_check_bool }
1083           {
1084             \zrefcheck_zcref_end_label_maybe:
1085             \zrefcheck_zcref_run_checks_on_labels:n
1086               { \l__zrefclever_zcref_labels_seq }
1087           }
1088      \group_end:
1089   }
```

(End definition for `\__zrefclever_zcref:nnnn`*.)*

```
1090 \seq_new:N \l__zrefclever_zcref_labels_seq
1091 \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

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

> `\zcpageref`⟨***⟩`[`⟨*options*⟩`]{`⟨*labels*⟩`}`

```
1092 \NewDocumentCommand \zcpageref { s O { } m }
1093   {
1094     \IfBooleanTF {#1}
1095       { \zcref*[#2, ref = page] {#3} }
1096       { \zcref [#2, ref = page] {#3} }
1097   }
```

(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.

```
1098 \tl_new:N \l__zrefclever_label_type_a_tl
1099 \tl_new:N \l__zrefclever_label_type_b_tl
1100 \tl_new:N \l__zrefclever_label_enclcnt_a_tl
1101 \tl_new:N \l__zrefclever_label_enclcnt_b_tl
1102 \tl_new:N \l__zrefclever_label_enclval_a_tl
1103 \tl_new:N \l__zrefclever_label_enclval_b_tl
```

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

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

```
1104 \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.

```
1105 \int_new:N \l__zrefclever_sort_prior_a_int
1106 \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.

```
1107 \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.

```
1108 \cs_new_protected:Npn \__zrefclever_sort_labels:
1109   {
```

Store label types sequence.

```
1110     \seq_clear:N \l__zrefclever_label_types_seq
1111     \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
1112       {
1113         \seq_map_function:NN \l__zrefclever_zcref_labels_seq
1114           \__zrefclever_label_type_put_new_right:n
1115       }
```

Sort.

```
1116     \seq_sort:Nn \l__zrefclever_zcref_labels_seq
1117       {
1118         \zref@ifrefundefined {##1}
1119           {
1120             \zref@ifrefundefined {##2}
1121               {
1122                 % Neither label is defined.
1123                 \sort_return_same:
1124               }
1125               {
1126                 % The second label is defined, but the first isn't, leave the
1127                 % undefined first (to be more visible).
1128                 \sort_return_same:
1129               }
1130           }
1131           {
1132             \zref@ifrefundefined {##2}
1133               {
1134                 % The first label is defined, but the second isn't, bring the
1135                 % second forward.
1136                 \sort_return_swapped:
```

```
1137                    }
1138                    {
1139                      % The interesting case: both labels are defined.  References
1140                      % to the "default" property or to the "page" are quite
1141                      % different with regard to sorting, so we branch them here to
1142                      % specialized functions.
1143                      \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1144                        { \__zrefclever_sort_page:nn {##1} {##2} }
1145                        { \__zrefclever_sort_default:nn {##1} {##2} }
1146                    }
1147                }
1148            }
1149      }
```

(*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*⟩}

```
1150 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
1151   {
1152     \tl_set:Nx \l__zrefclever_label_type_a_tl
1153       { \zref@extractdefault {#1} { zc@type } { \c_empty_tl } }
1154     \seq_if_in:NVF \l__zrefclever_label_types_seq
1155       \l__zrefclever_label_type_a_tl
1156       {
1157         \seq_put_right:NV \l__zrefclever_label_types_seq
1158           \l__zrefclever_label_type_a_tl
1159       }
1160   }
```

(*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*⟩}

```
1161 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
1162   {
1163     \tl_set:Nx \l__zrefclever_label_type_a_tl
1164       { \zref@extractdefault {#1} { zc@type } { \c_empty_tl } }
1165     \tl_set:Nx \l__zrefclever_label_type_b_tl
1166       { \zref@extractdefault {#2} { zc@type } { \c_empty_tl } }
```

```
1167
1168      \bool_if:nTF
1169        {
1170          % The second label has a type, but the first doesn't, leave the
1171          % undefined first (to be more visible).
1172          \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1173          ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1174        }
1175        { \sort_return_same: }
1176        {
1177          \bool_if:nTF
1178            {
1179              % The first label has a type, but the second doesn't, bring the
1180              % second forward.
1181              ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1182              \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1183            }
1184            { \sort_return_swapped: }
1185            {
1186              \bool_if:nTF
1187                {
1188                  % The interesting case: both labels have a type...
1189                  ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1190                  ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1191                }
1192                {
1193                  \tl_if_eq:NNTF
1194                    \l__zrefclever_label_type_a_tl
1195                    \l__zrefclever_label_type_b_tl
1196                    % ...and it's the same type.
1197                    { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
1198                    % ...and they are different types.
1199                    { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
1200                }
1201                {
1202                  % Neither label has a type.  We can't do much of meaningful
1203                  % here, but if it's the same counter, compare it.
1204                  \exp_args:Nxx \tl_if_eq:nnTF
1205                    { \zref@extractdefault {#1} { counter } { } }
1206                    { \zref@extractdefault {#2} { counter } { } }
1207                    {
1208                      \int_compare:nNnTF
1209                        { \zref@extractdefault {#1} { zc@cntval } { -1 } }
1210                          >
1211                        { \zref@extractdefault {#2} { zc@cntval } { -1 } }
1212                        { \sort_return_swapped: }
1213                        { \sort_return_same:    }
1214                    }
1215                    { \sort_return_same: }
1216                }
1217            }
1218        }
1219    }
```

(End definition for `\__zrefclever_sort_default:nn`*.)*

Variant not provided by the kernel, for use in `__zrefclever_sort_default_-
same_type:nn`.

```
1220 \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⟩}`

```
1221 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
1222   {
1223     \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1224       { \zref@extractdefault {#1} { zc@enclcnt } { \c_empty_tl } }
1225     \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1226       { \tl_reverse_items:V \l__zrefclever_label_enclcnt_a_tl }
1227     \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1228       { \zref@extractdefault {#2} { zc@enclcnt } { \c_empty_tl } }
1229     \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1230       { \tl_reverse_items:V \l__zrefclever_label_enclcnt_b_tl }
1231     \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1232       { \zref@extractdefault {#1} { zc@enclval } { \c_empty_tl } }
1233     \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1234       { \tl_reverse_items:V \l__zrefclever_label_enclval_a_tl }
1235     \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1236       { \zref@extractdefault {#2} { zc@enclval } { \c_empty_tl } }
1237     \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1238       { \tl_reverse_items:V \l__zrefclever_label_enclval_b_tl }
1239
1240     \bool_set_false:N \l__zrefclever_sort_decided_bool
1241     \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1242       {
1243         \bool_if:nTF
1244           {
1245             % Both are empty: neither label has any (further) "enclosing
1246             % counters" (left).
1247             \tl_if_empty_p:V \l__zrefclever_label_enclcnt_a_tl &&
1248             \tl_if_empty_p:V \l__zrefclever_label_enclcnt_b_tl
1249           }
1250           {
1251             \exp_args:Nxx \tl_if_eq:nnTF
1252               { \zref@extractdefault {#1} { counter } { } }
1253               { \zref@extractdefault {#2} { counter } { } }
1254               {
1255                 \bool_set_true:N \l__zrefclever_sort_decided_bool
1256                 \int_compare:nNnTF
1257                   { \zref@extractdefault {#1} { zc@cntval } { -1 } }
1258                     >
1259                   { \zref@extractdefault {#2} { zc@cntval } { -1 } }
1260                   { \sort_return_swapped: }
1261                   { \sort_return_same:     }
1262               }
1263               {
1264                 \msg_warning:nnnn { zref-clever }
1265                   { counters-not-nested } {#1} {#2}
1266                 \bool_set_true:N \l__zrefclever_sort_decided_bool
1267                 \sort_return_same:
1268               }
```

39

```
1269                    }
1270                    {
1271                      \bool_if:nTF
1272                        {
1273                          % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
1274                          \tl_if_empty_p:V \l__zrefclever_label_enclcnt_a_tl
1275                        }
1276                        {
1277                          \exp_args:NNx \tl_if_in:NnTF
1278                            \l__zrefclever_label_enclcnt_b_tl
1279                            { {\zref@extractdefault {#1} { counter } { }} }
1280                            {
1281                              \bool_set_true:N \l__zrefclever_sort_decided_bool
1282                              \sort_return_same:
1283                            }
1284                            {
1285                              \msg_warning:nnnn { zref-clever }
1286                                { counters-not-nested } {#1} {#2}
1287                              \bool_set_true:N \l__zrefclever_sort_decided_bool
1288                              \sort_return_same:
1289                            }
1290                        }
1291                        {
1292                          \bool_if:nTF
1293                            {
1294                              % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
1295                              \tl_if_empty_p:V \l__zrefclever_label_enclcnt_b_tl
1296                            }
1297                            {
1298                              \exp_args:NNx \tl_if_in:NnTF
1299                                \l__zrefclever_label_enclcnt_a_tl
1300                                { {\zref@extractdefault {#2} { counter } { }} }
1301                                {
1302                                  \bool_set_true:N \l__zrefclever_sort_decided_bool
1303                                  \sort_return_swapped:
1304                                }
1305                                {
1306                                  \msg_warning:nnnn { zref-clever }
1307                                    { counters-not-nested } {#1} {#2}
1308                                  \bool_set_true:N \l__zrefclever_sort_decided_bool
1309                                  \sort_return_same:
1310                                }
1311                            }
1312                            {
1313                              % Neither is empty: we can (possibly) compare the values
1314                              % of the current enclosing counter in the loop, if they
1315                              % are equal, we are still in the loop, if they are not, a
1316                              % sorting decision can be made directly.
1317                              \exp_args:Nxx \tl_if_eq:nnTF
1318                                { \tl_head:N \l__zrefclever_label_enclcnt_a_tl }
1319                                { \tl_head:N \l__zrefclever_label_enclcnt_b_tl }
1320                                {
1321                                  \int_compare:nNnTF
1322                                    { \tl_head:N \l__zrefclever_label_enclval_a_tl }
```

```
1323                                              =
1324                            { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1325                            {
1326                              \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1327                                { \tl_tail:N \l__zrefclever_label_enclcnt_a_tl }
1328                              \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1329                                { \tl_tail:N \l__zrefclever_label_enclcnt_b_tl }
1330                              \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1331                                { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1332                              \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1333                                { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1334                            }
1335                            {
1336                              \bool_set_true:N \l__zrefclever_sort_decided_bool
1337                              \int_compare:nNnTF
1338                                { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1339                                  >
1340                                { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1341                                { \sort_return_swapped: }
1342                                { \sort_return_same:    }
1343                            }
1344                        }
1345                        {
1346                          \msg_warning:nnnn { zref-clever }
1347                            { counters-not-nested } {#1} {#2}
1348                          \bool_set_true:N \l__zrefclever_sort_decided_bool
1349                          \sort_return_same:
1350                        }
1351                      }
1352                    }
1353                  }
1354              }
1355      }
```

(*End definition for* __zrefclever_sort_default_same_type:nn.)

__zrefclever_sort_default_different_types:nn {⟨*label a*⟩} {⟨*label b*⟩}

```
1356 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
1357    {
```

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".

```
1358      \int_zero:N \l__zrefclever_sort_prior_a_int
1359      \int_zero:N \l__zrefclever_sort_prior_b_int
1360      \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
1361        {
1362          \tl_if_eq:nnTF {##2} {{othertypes}}
1363            {
1364              \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
1365                { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1366              \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
1367                { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1368            }
```

```
1369              {
1370                \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
1371                  { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1372                  {
1373                    \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
1374                      { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1375                  }
1376              }
1377          }
```

Then do the actual sorting.

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

(*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*⟩}

```
1408 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
1409   {
1410     \int_compare:nNnTF
```

```
1411        { \zref@extractdefault {#1} { abspage } {-1} }
1412          >
1413        { \zref@extractdefault {#2} { abspage } {-1} }
1414        { \sort_return_swapped: }
1415        { \sort_return_same:    }
1416    }
```

(*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 do 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

`\l__zrefclever_typeset_labels_seq`
`\l__zrefclever_typeset_last_bool`
`\l__zrefclever_last_of_type_bool`

Auxiliary variables for `\__zrefclever_typeset_refs`: main stack control.

```
1417 \seq_new:N \l__zrefclever_typeset_labels_seq
1418 \bool_new:N \l__zrefclever_typeset_last_bool
1419 \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`.)

`\l__zrefclever_type_count_int`
`\l__zrefclever_label_count_int`

Auxiliary variables for `\__zrefclever_typeset_refs`: main counters.

```
1420 \int_new:N \l__zrefclever_type_count_int
1421 \int_new:N \l__zrefclever_label_count_int
```

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

`\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`

Auxiliary variables for `\__zrefclever_typeset_refs`: main "queue" control and storage.

```
1422 \tl_new:N \l__zrefclever_label_a_tl
1423 \tl_new:N \l__zrefclever_label_b_tl
1424 \tl_new:N \l__zrefclever_typeset_queue_prev_tl
1425 \tl_new:N \l__zrefclever_typeset_queue_curr_tl
1426 \tl_new:N \l__zrefclever_type_first_label_tl
```

```
1427 \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`

```
1428 \tl_new:N \l__zrefclever_type_name_tl
1429 \bool_new:N \l_zrefclever_name_in_link_bool
1430 \tl_new:N \l__zrefclever_name_format_tl
1431 \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`

```
1432 \int_new:N \l__zrefclever_range_count_int
1433 \int_new:N \l__zrefclever_range_same_count_int
1434 \tl_new:N \l__zrefclever_range_beg_label_tl
1435 \bool_new:N \l__zrefclever_next_maybe_range_bool
1436 \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`

```
1437 \tl_new:N \l__zrefclever_tpairsep_tl
1438 \tl_new:N \l__zrefclever_tlistsep_tl
1439 \tl_new:N \l__zrefclever_tlastsep_tl
1440 \tl_new:N \l__zrefclever_namesep_tl
1441 \tl_new:N \l__zrefclever_pairsep_tl
1442 \tl_new:N \l__zrefclever_listsep_tl
1443 \tl_new:N \l__zrefclever_lastsep_tl
1444 \tl_new:N \l__zrefclever_rangesep_tl
1445 \tl_new:N \l__zrefclever_refpre_out_tl
1446 \tl_new:N \l__zrefclever_refpos_out_tl
1447 \tl_new:N \l__zrefclever_refpre_in_tl
1448 \tl_new:N \l__zrefclever_refpos_in_tl
1449 \tl_new:N \l__zrefclever_namefont_tl
1450 \tl_new:N \l__zrefclever_reffont_out_tl
1451 \tl_new:N \l__zrefclever_reffont_in_tl
```

(*End definition for* `\l__zrefclever_tpairsep_tl` *and others.*)

Main functions

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

```
1452 \cs_new_protected:Npn \__zrefclever_typeset_refs:
1453   {
1454     \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
1455       \l__zrefclever_zcref_labels_seq
1456     \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
1457     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1458     \tl_clear:N \l__zrefclever_type_first_label_tl
1459     \tl_clear:N \l__zrefclever_type_first_label_type_tl
1460     \tl_clear:N \l__zrefclever_range_beg_label_tl
1461     \int_zero:N \l__zrefclever_label_count_int
```

```
1462        \int_zero:N \l__zrefclever_type_count_int
1463        \int_zero:N \l__zrefclever_range_count_int
1464        \int_zero:N \l__zrefclever_range_same_count_int
1465
1466        % Get type block options (not type-specific).
1467        \__zrefclever_get_ref_string:nN { tpairsep }
1468          \l__zrefclever_tpairsep_tl
1469        \__zrefclever_get_ref_string:nN { tlistsep }
1470          \l__zrefclever_tlistsep_tl
1471        \__zrefclever_get_ref_string:nN { tlastsep }
1472          \l__zrefclever_tlastsep_tl
1473
1474        % Process label stack.
1475        \bool_set_false:N \l__zrefclever_typeset_last_bool
1476        \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1477          {
1478            \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
1479              \l__zrefclever_label_a_tl
1480            \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
1481              {
1482                \tl_clear:N \l__zrefclever_label_b_tl
1483                \bool_set_true:N \l__zrefclever_typeset_last_bool
1484              }
1485              {
1486                \seq_get_left:NN \l__zrefclever_typeset_labels_seq
1487                  \l__zrefclever_label_b_tl
1488              }
1489
1490            \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1491              {
1492                \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1493                \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1494              }
1495              {
1496                \tl_set:Nx \l__zrefclever_label_type_a_tl
1497                  {
1498                    \zref@extractdefault
1499                      { \l__zrefclever_label_a_tl } { zc@type } { \c_empty_tl }
1500                  }
1501                \tl_set:Nx \l__zrefclever_label_type_b_tl
1502                  {
1503                    \zref@extractdefault
1504                      { \l__zrefclever_label_b_tl } { zc@type } { \c_empty_tl }
1505                  }
1506              }
1507
1508            % First, we establish whether the "current label" (i.e. 'a') is the
1509            % last one of its type.  This can happen because the "next label"
1510            % (i.e. 'b') is of a different type (or different definition status),
1511            % or because we are at the end of the list.
1512            \bool_if:NTF \l__zrefclever_typeset_last_bool
1513              { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1514              {
1515                \zref@ifrefundefined { \l__zrefclever_label_a_tl }
```

```
1516                    {
1517                      \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1518                        { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1519                        { \bool_set_true:N \l__zrefclever_last_of_type_bool  }
1520                    }
1521                    {
1522                      \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1523                        { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1524                        {
1525                          % Neither is undefined, we must check the types.
1526                          \bool_if:nTF
1527                            {
1528                              % Both empty: same "type".
1529                              \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1530                              \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1531                            }
1532                            { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1533                            {
1534                              \bool_if:nTF
1535                                {
1536                                  % Neither empty: compare types.
1537                                  ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl
1538                                  &&
1539                                  ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1540                                }
1541                                {
1542                                  \tl_if_eq:NNTF
1543                                    \l__zrefclever_label_type_a_tl
1544                                    \l__zrefclever_label_type_b_tl
1545                                    {
1546                                      \bool_set_false:N
1547                                        \l__zrefclever_last_of_type_bool
1548                                    }
1549                                    {
1550                                      \bool_set_true:N
1551                                        \l__zrefclever_last_of_type_bool
1552                                    }
1553                                }
1554                                % One empty, the other not: different "types".
1555                                {
1556                                  \bool_set_true:N
1557                                    \l__zrefclever_last_of_type_bool
1558                                }
1559                            }
1560                        }
1561                    }
1562              }

1564          % Handle warnings in case of reference or type undefined.
1565          \zref@refused { \l__zrefclever_label_a_tl }
1566          \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1567            {}
1568            {
1569              \tl_if_empty:NT \l__zrefclever_label_type_a_tl
```

47

```
1570                 {
1571                   \msg_warning:nnx { zref-clever } { missing-type }
1572                     { \l__zrefclever_label_a_tl }
1573                 }
1574             }
1575
1576         % Get type-specific separators, refpre/pos and font options, once per
1577         % type.
1578         \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
1579           {
1580             \__zrefclever_get_ref_string:nN { namesep    }
1581               \l__zrefclever_namesep_tl
1582             \__zrefclever_get_ref_string:nN { rangesep   }
1583               \l__zrefclever_rangesep_tl
1584             \__zrefclever_get_ref_string:nN { pairsep    }
1585               \l__zrefclever_pairsep_tl
1586             \__zrefclever_get_ref_string:nN { listsep    }
1587               \l__zrefclever_listsep_tl
1588             \__zrefclever_get_ref_string:nN { lastsep    }
1589               \l__zrefclever_lastsep_tl
1590             \__zrefclever_get_ref_string:nN { refpre     }
1591               \l__zrefclever_refpre_out_tl
1592             \__zrefclever_get_ref_string:nN { refpos     }
1593               \l__zrefclever_refpos_out_tl
1594             \__zrefclever_get_ref_string:nN { refpre-in  }
1595               \l__zrefclever_refpre_in_tl
1596             \__zrefclever_get_ref_string:nN { refpos-in  }
1597               \l__zrefclever_refpos_in_tl
1598             \__zrefclever_get_ref_font:nN   { namefont   }
1599               \l__zrefclever_namefont_tl
1600             \__zrefclever_get_ref_font:nN   { reffont    }
1601               \l__zrefclever_reffont_out_tl
1602             \__zrefclever_get_ref_font:nN   { reffont-in }
1603               \l__zrefclever_reffont_in_tl
1604           }
1605
1606         % Here we send this to a couple of auxiliary functions.
1607         \bool_if:NTF \l__zrefclever_last_of_type_bool
1608           % There exists no next label of the same type as the current.
1609           { \__zrefclever_typeset_refs_last_of_type: }
1610           % There exists a next label of the same type as the current.
1611           { \__zrefclever_typeset_refs_not_last_of_type: }
1612       }
1613   }
```

(*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_-

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

```
1614 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
1615   {
1616     % Process the current label to the current queue.
1617     \int_case:nnF { \l__zrefclever_label_count_int }
1618       {
1619         % It is the last label of its type, but also the first one, and that's
1620         % what matters here: just store it.
1621         { 0 }
1622         {
1623           \tl_set:NV \l__zrefclever_type_first_label_tl
1624             \l__zrefclever_label_a_tl
1625           \tl_set:NV \l__zrefclever_type_first_label_type_tl
1626             \l__zrefclever_label_type_a_tl
1627         }
1628
1629         % The last is the second: we have a pair (if not repeated).
1630         { 1 }
1631         {
1632           \int_compare:nNnF { \l__zrefclever_range_same_count_int } = { 1 }
1633             {
1634               \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1635                 {
1636                   \exp_not:V \l__zrefclever_pairsep_tl
1637                   \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1638                 }
1639             }
1640         }
1641       }
1642     % Last is third or more of its type: without repetition, we'd have the
1643     % last element on a list, but control for possible repetition.
1644       {
1645         \int_case:nnF { \l__zrefclever_range_count_int }
1646           {
1647             % There was no range going on.
1648             { 0 }
1649             {
1650               \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1651                 {
1652                   \exp_not:V \l__zrefclever_lastsep_tl
1653                   \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1654                 }
1655             }
1656             % Last in the range is also the second in it.
1657             { 1 }
1658             {
1659               \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1660                 {
1661                   % We know 'range_beg_label' is not empty, since this is the
1662                   % second element in the range, but the third or more in the
1663                   % type list.
1664                   \exp_not:V \l__zrefclever_listsep_tl
```

```
1665                    \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1666                    \int_compare:nNnF
1667                      { \l__zrefclever_range_same_count_int } = { 1 }
1668                      {
1669                        \exp_not:V \l__zrefclever_lastsep_tl
1670                        \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1671                      }
1672                  }
1673              }
1674          }
1675          % Last in the range is third or more in it.
1676          {
1677            \int_case:nnF
1678              {
1679                \l__zrefclever_range_count_int -
1680                \l__zrefclever_range_same_count_int
1681              }
1682              {
1683                % Repetition, not a range.
1684                { 0 }
1685                {
1686                  % If 'range_beg_label' is empty, it means it was also the
1687                  % first of the type, and hence was already handled.
1688                  \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1689                    {
1690                      \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1691                        {
1692                          \exp_not:V \l__zrefclever_lastsep_tl
1693                          \__zrefclever_get_ref:V
1694                            \l__zrefclever_range_beg_label_tl
1695                        }
1696                    }
1697                }
1698                % A 'range', but with no skipped value, treat as list.
1699                { 1 }
1700                {
1701                  \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1702                    {
1703                      % Ditto.
1704                      \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1705                        {
1706                          \exp_not:V \l__zrefclever_listsep_tl
1707                          \__zrefclever_get_ref:V
1708                            \l__zrefclever_range_beg_label_tl
1709                        }
1710                      \exp_not:V \l__zrefclever_lastsep_tl
1711                      \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1712                    }
1713                }
1714              }
1715              {
1716                % An actual range.
1717                \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1718                  {
```

```
1719                       % Ditto.
1720                       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1721                         {
1722                           \exp_not:V \l__zrefclever_lastsep_tl
1723                           \__zrefclever_get_ref:V
1724                             \l__zrefclever_range_beg_label_tl
1725                         }
1726                       \exp_not:V \l__zrefclever_rangesep_tl
1727                       \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1728                     }
1729                 }
1730             }
1731         }
1732
1733     % Handle "range" option.  The idea is simple: if the queue is not empty,
1734     % we replace it with the end of the range (or pair).  We can still
1735     % retrieve the end of the range from 'label_a' since we know to be
1736     % processing the last label of its type at this point.
1737     \bool_if:NT \l__zrefclever_typeset_range_bool
1738       {
1739         \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
1740           {
1741             \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1742               { }
1743               {
1744                 \msg_warning:nnx { zref-clever } { single-element-range }
1745                   { \l__zrefclever_type_first_label_type_tl }
1746               }
1747           }
1748           {
1749             \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1750             \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1751               { }
1752               {
1753                 \__zrefclever_labels_in_sequence:nn
1754                   { \l__zrefclever_type_first_label_tl }
1755                   { \l__zrefclever_label_a_tl }
1756               }
1757             \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1758               {
1759                 \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1760                   { \exp_not:V \l__zrefclever_pairsep_tl }
1761                   { \exp_not:V \l__zrefclever_rangesep_tl }
1762                 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1763               }
1764           }
1765       }
1766
1767     % Now that the type block is finished, we can add the name and the first
1768     % ref to the queue.  Also, if "typeset" option is not "both", handle it
1769     % here as well.
1770     \__zrefclever_type_name_setup:
1771     \bool_if:nTF
1772       { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
```

51

```
1773          {
1774            \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1775              { \__zrefclever_get_ref_first: }
1776          }
1777          {
1778            \bool_if:nTF
1779              { \l__zrefclever_typeset_ref_bool }
1780              {
1781                \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1782                  { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
1783              }
1784              {
1785                \bool_if:nTF
1786                  { \l__zrefclever_typeset_name_bool }
1787                  {
1788                    \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1789                      {
1790                        \bool_if:NTF \l__zrefclever_name_in_link_bool
1791                          {
1792                            \exp_not:N \group_begin:
1793                            \exp_not:V \l__zrefclever_namefont_tl
1794                            % It's two '@s', but escaped for DocStrip.
1795                            \exp_not:N \hyper@@link
1796                              {
1797                                \zref@ifrefcontainsprop
1798                                  { \l__zrefclever_type_first_label_tl }
1799                                  { urluse }
1800                                  {
1801                                    \zref@extractdefault
1802                                      { \l__zrefclever_type_first_label_tl }
1803                                      { urluse } {}
1804                                  }
1805                                  {
1806                                    \zref@extractdefault
1807                                      { \l__zrefclever_type_first_label_tl }
1808                                      { url } {}
1809                                  }
1810                              }
1811                              {
1812                                \zref@extractdefault
1813                                  { \l__zrefclever_type_first_label_tl }
1814                                  { anchor } {}
1815                              }
1816                              { \exp_not:V \l__zrefclever_type_name_tl }
1817                            \exp_not:N \group_end:
1818                          }
1819                          {
1820                            \exp_not:N \group_begin:
1821                            \exp_not:V \l__zrefclever_namefont_tl
1822                            \exp_not:V \l__zrefclever_type_name_tl
1823                            \exp_not:N \group_end:
1824                          }
1825                      }
1826                  }
```

```
1827                {
1828                  % Logically, this case would correspond to "typeset=none", but
1829                  % it should not occur, given that the options are set up to
1830                  % typeset either "ref" or "name".  Still, leave here a
1831                  % sensible fallback, equal to the behavior of "both".
1832                  \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1833                    { \__zrefclever_get_ref_first: }
1834                }
1835            }
1836        }
1837
1838    % Typeset the previous type, if there is one.
1839    \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
1840      {
1841        \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
1842          { \l__zrefclever_tlistsep_tl }
1843        \l__zrefclever_typeset_queue_prev_tl
1844      }
1845
1846    % Wrap up loop, or prepare for next iteration.
1847    \bool_if:NTF \l__zrefclever_typeset_last_bool
1848      {
1849        % We are finishing, typeset the current queue.
1850        \int_case:nnF { \l__zrefclever_type_count_int }
1851          {
1852            % Single type.
1853            { 0 }
1854            { \l__zrefclever_typeset_queue_curr_tl }
1855            % Pair of types.
1856            { 1 }
1857            {
1858              \l__zrefclever_tpairsep_tl
1859              \l__zrefclever_typeset_queue_curr_tl
1860            }
1861          }
1862          {
1863            % Last in list of types.
1864            \l__zrefclever_tlastsep_tl
1865            \l__zrefclever_typeset_queue_curr_tl
1866          }
1867      }
1868      {
1869        % There are further labels, set variables for next iteration.
1870        \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
1871          \l__zrefclever_typeset_queue_curr_tl
1872        \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1873        \tl_clear:N \l__zrefclever_type_first_label_tl
1874        \tl_clear:N \l__zrefclever_type_first_label_type_tl
1875        \tl_clear:N \l__zrefclever_range_beg_label_tl
1876        \int_zero:N \l__zrefclever_label_count_int
1877        \int_incr:N \l__zrefclever_type_count_int
1878        \int_zero:N \l__zrefclever_range_count_int
1879        \int_zero:N \l__zrefclever_range_same_count_int
1880      }
```

53

```
1881    }
```

(*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.

```
1882  \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
1883    {
1884      % Signal if next label may form a range with the current one (only
1885      % considered if compression is enabled in the first place).
1886      \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1887      \bool_set_false:N \l__zrefclever_next_is_same_bool
1888      \bool_if:NT \l__zrefclever_typeset_compress_bool
1889        {
1890          \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1891            { }
1892            {
1893              \__zrefclever_labels_in_sequence:nn
1894                { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
1895            }
1896        }
1897
1898      % Process the current label to the current queue.
1899      \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
1900        {
1901          % Current label is the first of its type (also not the last, but it
1902          % doesn't matter here): just store the label.
1903          \tl_set:NV \l__zrefclever_type_first_label_tl
1904            \l__zrefclever_label_a_tl
1905          \tl_set:NV \l__zrefclever_type_first_label_type_tl
1906            \l__zrefclever_label_type_a_tl
1907
1908          % If the next label may be part of a range, we set 'range_beg_label'
1909          % to "empty" (we deal with it as the "first", and must do it there, to
1910          % handle hyperlinking), but also step the range counters.
1911          \bool_if:NT \l__zrefclever_next_maybe_range_bool
1912            {
1913              \tl_clear:N \l__zrefclever_range_beg_label_tl
1914              \int_incr:N \l__zrefclever_range_count_int
1915              \bool_if:NT \l__zrefclever_next_is_same_bool
1916                { \int_incr:N \l__zrefclever_range_same_count_int }
1917            }
1918        }
1919        {
1920          % Current label is neither the first (nor the last) of its type.
1921          \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1922            {
1923              % Starting, or continuing a range.
1924              \int_compare:nNnTF
1925                { \l__zrefclever_range_count_int } = { 0 }
1926                {
1927                  % There was no range going, we are starting one.
1928                  \tl_set:NV \l__zrefclever_range_beg_label_tl
1929                    \l__zrefclever_label_a_tl
1930                  \int_incr:N \l__zrefclever_range_count_int
```

```
1931          \bool_if:NT \l__zrefclever_next_is_same_bool
1932            { \int_incr:N \l__zrefclever_range_same_count_int }
1933        }
1934        {
1935          % Second or more in the range, but not the last.
1936          \int_incr:N \l__zrefclever_range_count_int
1937          \bool_if:NT \l__zrefclever_next_is_same_bool
1938            { \int_incr:N \l__zrefclever_range_same_count_int }
1939        }
1940    }
1941    {
1942      % Next element is not in sequence: there was no range, or we are
1943      % closing one.
1944      \int_case:nnF { \l__zrefclever_range_count_int }
1945        {
1946          % There was no range going on.
1947          { 0 }
1948          {
1949            \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1950              {
1951                \exp_not:V \l__zrefclever_listsep_tl
1952                \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1953              }
1954          }
1955          % Last is second in the range: if 'range_same_count' is also
1956          % '1', it's a repetition (drop it), otherwise, it's a "pair
1957          % within a list", treat as list.
1958          { 1 }
1959          {
1960            \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1961              {
1962                \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1963                  {
1964                    \exp_not:V \l__zrefclever_listsep_tl
1965                    \__zrefclever_get_ref:V
1966                      \l__zrefclever_range_beg_label_tl
1967                  }
1968                \int_compare:nNnF
1969                  { \l__zrefclever_range_same_count_int } = { 1 }
1970                  {
1971                    \exp_not:V \l__zrefclever_listsep_tl
1972                    \__zrefclever_get_ref:V
1973                      \l__zrefclever_label_a_tl
1974                  }
1975              }
1976          }
1977        }
1978        {
1979          % Last is third or more in the range: if 'range_count' and
1980          % 'range_same_count' are the same, its a repetition (drop it),
1981          % if they differ by '1', its a list, if they differ by more,
1982          % it is a real range.
1983          \int_case:nnF
1984            {
```

```
1985                      \l__zrefclever_range_count_int -
1986                      \l__zrefclever_range_same_count_int
1987                    }
1988                    {
1989                      { 0 }
1990                      {
1991                        \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1992                          {
1993                            \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1994                              {
1995                                \exp_not:V \l__zrefclever_listsep_tl
1996                                \__zrefclever_get_ref:V
1997                                  \l__zrefclever_range_beg_label_tl
1998                              }
1999                          }
2000                      }
2001                      { 1 }
2002                      {
2003                        \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2004                          {
2005                            \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2006                              {
2007                                \exp_not:V \l__zrefclever_listsep_tl
2008                                \__zrefclever_get_ref:V
2009                                  \l__zrefclever_range_beg_label_tl
2010                              }
2011                            \exp_not:V \l__zrefclever_listsep_tl
2012                            \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2013                          }
2014                      }
2015                    }
2016                    {
2017                      \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2018                        {
2019                          \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2020                            {
2021                              \exp_not:V \l__zrefclever_listsep_tl
2022                              \__zrefclever_get_ref:V
2023                                \l__zrefclever_range_beg_label_tl
2024                            }
2025                          \exp_not:V \l__zrefclever_rangesep_tl
2026                          \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2027                        }
2028                    }
2029                  }
2030              % Reset counters.
2031              \int_zero:N \l__zrefclever_range_count_int
2032              \int_zero:N \l__zrefclever_range_same_count_int
2033            }
2034        }
2035      % Step label counter for next iteration.
2036      \int_incr:N \l__zrefclever_label_count_int
2037    }
```

(*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 TeXnical 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:`
`\__zrefclever_name_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.

```
2038 \cs_new_protected:Npn \__zrefclever_ref_default:
2039   { \zref@default }
2040 \cs_new_protected:Npn \__zrefclever_name_default:
2041   { \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⟩}`

```
2042 \cs_new:Npn \__zrefclever_get_ref:n #1
2043   {
2044     \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
2045       {
2046         \bool_if:nTF
2047           {
2048             \l__zrefclever_use_hyperref_bool &&
2049             ! \l__zrefclever_link_star_bool
2050           }
2051           {
2052             \exp_not:N \group_begin:
2053             \exp_not:V \l__zrefclever_reffont_out_tl
2054             \exp_not:V \l__zrefclever_refpre_out_tl
```

57

```
2055              \exp_not:N \group_begin:
2056              \exp_not:V \l__zrefclever_reffont_in_tl
2057              % It's two '@s', but escaped for DocStrip.
2058              \exp_not:N \hyper@@link
2059                {
2060                  \zref@ifrefcontainsprop {#1} { urluse }
2061                    { \zref@extractdefault {#1} { urluse } { } }
2062                    { \zref@extractdefault {#1} { url } { } }
2063                }
2064                { \zref@extractdefault {#1} { anchor } { } }
2065                {
2066                  \exp_not:V \l__zrefclever_refpre_in_tl
2067                  \zref@extractdefault {#1}
2068                    { \l__zrefclever_ref_property_tl } { }
2069                  \exp_not:V \l__zrefclever_refpos_in_tl
2070                }
2071              \exp_not:N \group_end:
2072              \exp_not:V \l__zrefclever_refpos_out_tl
2073              \exp_not:N \group_end:
2074            }
2075            {
2076              \exp_not:N \group_begin:
2077              \exp_not:V \l__zrefclever_reffont_out_tl
2078              \exp_not:V \l__zrefclever_refpre_out_tl
2079              \exp_not:N \group_begin:
2080              \exp_not:V \l__zrefclever_reffont_in_tl
2081              \exp_not:V \l__zrefclever_refpre_in_tl
2082              \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } { }
2083              \exp_not:V \l__zrefclever_refpos_in_tl
2084              \exp_not:N \group_end:
2085              \exp_not:V \l__zrefclever_refpos_out_tl
2086              \exp_not:N \group_end:
2087            }
2088        }
2089        { \__zrefclever_ref_default: }
2090    }
2091 \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.

```
2092 \cs_new:Npn \__zrefclever_get_ref_first:
2093    {
2094      \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2095        { \__zrefclever_ref_default: }
2096        {
2097          \bool_if:NTF \l__zrefclever_name_in_link_bool
```

58

```
2098            {
2099              \zref@ifrefcontainsprop
2100                { \l__zrefclever_type_first_label_tl }
2101                { \l__zrefclever_ref_property_tl }
2102                {
2103                  % It's two '@s', but escaped for DocStrip.
2104                  \exp_not:N \hyper@@link
2105                    {
2106                      \zref@ifrefcontainsprop
2107                        { \l__zrefclever_type_first_label_tl } { urluse }
2108                        {
2109                          \zref@extractdefault
2110                            { \l__zrefclever_type_first_label_tl }
2111                            { urluse } { }
2112                        }
2113                        {
2114                          \zref@extractdefault
2115                            { \l__zrefclever_type_first_label_tl }
2116                            { url } { }
2117                        }
2118                    }
2119                    {
2120                      \zref@extractdefault
2121                        { \l__zrefclever_type_first_label_tl }
2122                        { anchor } { }
2123                    }
2124                    {
2125                      \exp_not:N \group_begin:
2126                      \exp_not:V \l__zrefclever_namefont_tl
2127                      \exp_not:V \l__zrefclever_type_name_tl
2128                      \exp_not:N \group_end:
2129                      \exp_not:V \l__zrefclever_namesep_tl
2130                      \exp_not:N \group_begin:
2131                      \exp_not:V \l__zrefclever_reffont_out_tl
2132                      \exp_not:V \l__zrefclever_refpre_out_tl
2133                      \exp_not:N \group_begin:
2134                      \exp_not:V \l__zrefclever_reffont_in_tl
2135                      \exp_not:V \l__zrefclever_refpre_in_tl
2136                      \zref@extractdefault
2137                        { \l__zrefclever_type_first_label_tl }
2138                        { \l__zrefclever_ref_property_tl } { }
2139                      \exp_not:V \l__zrefclever_refpos_in_tl
2140                      \exp_not:N \group_end:
2141                      % hyperlink makes it's own group, we'd like to close the
2142                      % 'refpre-out' group after 'refpos-out', but... we close
2143                      % it here, and give the trailing 'refpos-out' its own
2144                      % group.  This will result that formatting given to
2145                      % 'refpre-out' will not reach 'refpos-out', but I see no
2146                      % alternative, and this has to be handled specially.
2147                      \exp_not:N \group_end:
2148                    }
2149              \exp_not:N \group_begin:
2150              % Ditto: special treatment.
2151              \exp_not:V \l__zrefclever_reffont_out_tl
```

```
2152                  \exp_not:V \l__zrefclever_refpos_out_tl
2153                  \exp_not:N \group_end:
2154                }
2155                {
2156                  \exp_not:N \group_begin:
2157                  \exp_not:V \l__zrefclever_namefont_tl
2158                  \exp_not:V \l__zrefclever_type_name_tl
2159                  \exp_not:N \group_end:
2160                  \exp_not:V \l__zrefclever_namesep_tl
2161                  \__zrefclever_ref_default:
2162                }
2163            }
2164            {
2165              \tl_if_empty:NTF \l__zrefclever_type_name_tl
2166                {
2167                  \__zrefclever_name_default:
2168                  \exp_not:V \l__zrefclever_namesep_tl
2169                }
2170                {
2171                  \exp_not:N \group_begin:
2172                  \exp_not:V \l__zrefclever_namefont_tl
2173                  \exp_not:V \l__zrefclever_type_name_tl
2174                  \exp_not:N \group_end:
2175                  \exp_not:V \l__zrefclever_namesep_tl
2176                }
2177              \zref@ifrefcontainsprop
2178                { \l__zrefclever_type_first_label_tl }
2179                { \l__zrefclever_ref_property_tl }
2180                {
2181                  \bool_if:nTF
2182                    {
2183                      \l__zrefclever_use_hyperref_bool &&
2184                      ! \l__zrefclever_link_star_bool
2185                    }
2186                    {
2187                      \exp_not:N \group_begin:
2188                      \exp_not:V \l__zrefclever_reffont_out_tl
2189                      \exp_not:V \l__zrefclever_refpre_out_tl
2190                      \exp_not:N \group_begin:
2191                      \exp_not:V \l__zrefclever_reffont_in_tl
2192                      % It's two '@s', but escaped for DocStrip.
2193                      \exp_not:N \hyper@@link
2194                        {
2195                          \zref@ifrefcontainsprop
2196                            { \l__zrefclever_type_first_label_tl } { urluse }
2197                            {
2198                              \zref@extractdefault
2199                                { \l__zrefclever_type_first_label_tl }
2200                                { urluse } { }
2201                            }
2202                            {
2203                              \zref@extractdefault
2204                                { \l__zrefclever_type_first_label_tl }
2205                                { url } { }
```

60

```
2206                             }
2207                           }
2208                           {
2209                             \zref@extractdefault
2210                               { \l__zrefclever_type_first_label_tl }
2211                               { anchor } { }
2212                           }
2213                           {
2214                             \exp_not:V \l__zrefclever_refpre_in_tl
2215                             \zref@extractdefault
2216                               { \l__zrefclever_type_first_label_tl }
2217                               { \l__zrefclever_ref_property_tl } { }
2218                             \exp_not:V \l__zrefclever_refpos_in_tl
2219                           }
2220                         \exp_not:N \group_end:
2221                         \exp_not:V \l__zrefclever_refpos_out_tl
2222                         \exp_not:N \group_end:
2223                       }
2224                       {
2225                         \exp_not:N \group_begin:
2226                         \exp_not:V \l__zrefclever_reffont_out_tl
2227                         \exp_not:V \l__zrefclever_refpre_out_tl
2228                         \exp_not:N \group_begin:
2229                         \exp_not:V \l__zrefclever_reffont_in_tl
2230                         \exp_not:V \l__zrefclever_refpre_in_tl
2231                         \zref@extractdefault
2232                           { \l__zrefclever_type_first_label_tl }
2233                           { \l__zrefclever_ref_property_tl } { }
2234                         \exp_not:V \l__zrefclever_refpos_in_tl
2235                         \exp_not:N \group_end:
2236                         \exp_not:V \l__zrefclever_refpos_out_tl
2237                         \exp_not:N \group_end:
2238                       }
2239                   }
2240                 { \__zrefclever_ref_default: }
2241             }
2242         }
2243     }
```

(*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.

```
2244 \cs_new_protected:Npn \__zrefclever_type_name_setup:
```

```
2245    {
2246      \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2247        { \tl_clear:N \l__zrefclever_type_name_tl }
2248        {
2249          \tl_if_empty:nTF \l__zrefclever_type_first_label_type_tl
2250            { \tl_clear:N \l__zrefclever_type_name_tl }
2251            {
2252              % Determine whether we should use capitalization, abbreviation,
2253              % and plural.
2254              \bool_lazy_or:nnTF
2255                { \l__zrefclever_capitalize_bool }
2256                {
2257                  \l__zrefclever_capitalize_first_bool &&
2258                  \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2259                }
2260                { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
2261                { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
2262              % If the queue is empty, we have a singular, otherwise, plural.
2263              \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2264                { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
2265                { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
2266              \bool_lazy_and:nnTF
2267                { \l__zrefclever_abbrev_bool }
2268                {
2269                  ! \int_compare_p:nNn
2270                     { \l__zrefclever_type_count_int } = { 0 } ||
2271                  ! \l__zrefclever_noabbrev_first_bool
2272                }
2273                {
2274                  \tl_set:NV \l__zrefclever_name_format_fallback_tl
2275                    \l__zrefclever_name_format_tl
2276                  \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
2277                }
2278                { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
2279
2280              \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
2281                {
2282                  \prop_get:cVNF
2283                    {
2284                      l__zrefclever_type_
2285                      \l__zrefclever_type_first_label_type_tl _options_prop
2286                    }
2287                    \l__zrefclever_name_format_tl
2288                    \l__zrefclever_type_name_tl
2289                    {
2290                      \__zrefclever_get_type_transl:xxxNF
2291                        { \l__zrefclever_ref_language_tl }
2292                        { \l__zrefclever_type_first_label_type_tl }
2293                        { \l__zrefclever_name_format_tl }
2294                        \l__zrefclever_type_name_tl
2295                        {
2296                          \tl_clear:N \l__zrefclever_type_name_tl
2297                          \msg_warning:nnx { zref-clever } { missing-name }
2298                            { \l__zrefclever_type_first_label_type_tl }
```

```
2299                              }
2300                            }
2301                          }
2302                          {
2303                            \prop_get:cVNF
2304                              {
2305                                l__zrefclever_type_
2306                                \l__zrefclever_type_first_label_type_tl _options_prop
2307                              }
2308                            \l__zrefclever_name_format_tl
2309                            \l__zrefclever_type_name_tl
2310                            {
2311                              \prop_get:cVNF
2312                                {
2313                                  l__zrefclever_type_
2314                                  \l__zrefclever_type_first_label_type_tl _options_prop
2315                                }
2316                              \l__zrefclever_name_format_fallback_tl
2317                              \l__zrefclever_type_name_tl
2318                              {
2319                                \__zrefclever_get_type_transl:xxxNF
2320                                  { \l__zrefclever_ref_language_tl }
2321                                  { \l__zrefclever_type_first_label_type_tl }
2322                                  { \l__zrefclever_name_format_tl }
2323                                  \l__zrefclever_type_name_tl
2324                                  {
2325                                    \__zrefclever_get_type_transl:xxxNF
2326                                      { \l__zrefclever_ref_language_tl }
2327                                      { \l__zrefclever_type_first_label_type_tl }
2328                                      { \l__zrefclever_name_format_fallback_tl }
2329                                      \l__zrefclever_type_name_tl
2330                                      {
2331                                        \tl_clear:N \l__zrefclever_type_name_tl
2332                                        \msg_warning:nnx { zref-clever }
2333                                          { missing-name }
2334                                          { \l__zrefclever_type_first_label_type_tl }
2335                                      }
2336                                  }
2337                              }
2338                            }
2339                          }
2340                        }
2341                    }
2342
2343      % Signal whether the type name is to be included in the hyperlink or not.
2344      \bool_lazy_any:nTF
2345        {
2346          { ! \l__zrefclever_use_hyperref_bool }
2347          { \l__zrefclever_link_star_bool }
2348          { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
2349          { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
2350        }
2351        { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2352        {
```

63

```
2353          \bool_lazy_any:nTF
2354            {
2355              { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
2356              {
2357                \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
2358                \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2359              }
2360              {
2361                \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
2362                \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2363                \l__zrefclever_typeset_last_bool &&
2364                \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2365              }
2366            }
2367            { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2368            { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2369        }
2370    }
```

(*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 ⟨*label b*⟩ comes in immediate sequence
from ⟨*label a*⟩. And sets both \l__zrefclever_next_maybe_range_bool and \l__-
zrefclever_next_is_same_bool to true if the two labels are the "same" (that is,
have the same counter value). These two boolean variables are the basis for all range
and compression handling inside __zrefclever_typeset_refs_not_last_of_type:,
so this function is expected to be called at its beginning, if compression is enabled.

         __zrefclever_labels_in_sequence:nn {⟨*label a*⟩} {⟨*label b*⟩}

```
2371  \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
2372    {
2373      \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2374        {
2375          \exp_args:Nxx \tl_if_eq:nnT
2376            { \zref@extractdefault {#1} { zc@pgfmt } { } }
2377            { \zref@extractdefault {#2} { zc@pgfmt } { } }
2378            {
2379              \int_compare:nNnTF
2380                { \zref@extractdefault {#1} { zc@pgval } { -2 } + 1 }
2381                  =
2382                { \zref@extractdefault {#2} { zc@pgval } { -1 } }
2383                { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2384                {
2385                  \int_compare:nNnT
2386                    { \zref@extractdefault {#1} { zc@pgval } { -1 } }
2387                      =
2388                    { \zref@extractdefault {#2} { zc@pgval } { -1 } }
2389                    {
2390                      \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2391                      \bool_set_true:N \l__zrefclever_next_is_same_bool
2392                    }
2393                }
2394            }
```

```
2395            }
2396            {
2397              \exp_args:Nxx \tl_if_eq:nnT
2398                { \zref@extractdefault {#1} { counter } { } }
2399                { \zref@extractdefault {#2} { counter } { } }
2400                {
2401                  \exp_args:Nxx \tl_if_eq:nnT
2402                    { \zref@extractdefault {#1} { zc@enclval } { } }
2403                    { \zref@extractdefault {#2} { zc@enclval } { } }
2404                    {
2405                      \int_compare:nNnTF
2406                        { \zref@extractdefault {#1} { zc@cntval } { -2 } + 1 }
2407                          =
2408                        { \zref@extractdefault {#2} { zc@cntval } { -1 } }
2409                        { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2410                        {
2411                          \int_compare:nNnT
2412                            { \zref@extractdefault {#1} { zc@cntval } { -1 } }
2413                              =
2414                            { \zref@extractdefault {#2} { zc@cntval } { -1 } }
2415                            {
2416                              \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2417                              \bool_set_true:N \l__zrefclever_next_is_same_bool
2418                            }
2419                        }
2420                    }
2421                }
2422            }
2423        }
```

(*End definition for* `\__zrefclever_labels_in_sequence:nn`.)

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

`\__zrefclever_get_ref_string:nN`   `\__zrefclever_get_ref_string:nN {`⟨*option*⟩`} {`⟨*tl variable*⟩`}`

```
2424 \cs_new_protected:Npn \__zrefclever_get_ref_string:nN #1#2
2425   {
2426     % First attempt: general options.
2427     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2428       {
2429         % If not found, try type specific options.
2430         \bool_lazy_all:nTF
```

65

```
2431                 {
2432                   { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2433                   {
2434                     \prop_if_exist_p:c
2435                       {
2436                         l__zrefclever_type_
2437                         \l__zrefclever_label_type_a_tl _options_prop
2438                       }
2439                   }
2440                   {
2441                     \prop_if_in_p:cn
2442                       {
2443                         l__zrefclever_type_
2444                         \l__zrefclever_label_type_a_tl _options_prop
2445                       }
2446                       {#1}
2447                   }
2448                 }
2449                 {
2450                   \prop_get:cnN
2451                     {
2452                       l__zrefclever_type_
2453                       \l__zrefclever_label_type_a_tl _options_prop
2454                     }
2455                     {#1} #2
2456                 }
2457                 {
2458                   % If not found, try type specific translations.
2459                   \__zrefclever_get_type_transl:xxnNF
2460                     { \l__zrefclever_ref_language_tl }
2461                     { \l__zrefclever_label_type_a_tl }
2462                     {#1} #2
2463                     {
2464                       % If not found, try default translations.
2465                       \__zrefclever_get_default_transl:xnNF
2466                         { \l__zrefclever_ref_language_tl }
2467                         {#1} #2
2468                         {
2469                           % If not found, try fallback.
2470                           \__zrefclever_get_fallback_transl:nNF {#1} #2
2471                             {
2472                               \tl_clear:N #2
2473                               \msg_warning:nnn { zref-clever }
2474                                 { missing-string } {#1}
2475                             }
2476                         }
2477                     }
2478                 }
2479           }
2480       }
```

(End definition for `\__zrefclever_get_ref_string:nN`.*)*

`\__zrefclever_get_ref_font:nN`   `\__zrefclever_get_ref_font:nN` {⟨*option*⟩} {⟨*tl variable*⟩}

```
2481  \cs_new_protected:Npn \__zrefclever_get_ref_font:nN #1#2
2482    {
2483      % First attempt: general options.
2484      \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2485        {
2486          % If not found, try type specific options.
2487          \bool_lazy_and:nnTF
2488            { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2489            {
2490              \prop_if_exist_p:c
2491                {
2492                  l__zrefclever_type_
2493                  \l__zrefclever_label_type_a_tl _options_prop
2494                }
2495            }
2496            {
2497              \prop_get:cnNF
2498                {
2499                  l__zrefclever_type_
2500                  \l__zrefclever_label_type_a_tl _options_prop
2501                }
2502                {#1} #2
2503                { \tl_clear:N #2 }
2504            }
2505            { \tl_clear:N #2 }
2506        }
2507    }
```

(*End definition for* `\__zrefclever_get_ref_font:nN`.)

9 Special handling

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

9.1 \appendix

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

9.2 **enumitem** package

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

2508 ⟨/package⟩

10 Dictionaries

10.1 English

```
2509 ⟨package⟩\zcDeclareLanguage { english }
2510 ⟨package⟩\zcDeclareLanguageAlias { american   } { english }
2511 ⟨package⟩\zcDeclareLanguageAlias { australian } { english }
2512 ⟨package⟩\zcDeclareLanguageAlias { british    } { english }
2513 ⟨package⟩\zcDeclareLanguageAlias { canadian   } { english }
2514 ⟨package⟩\zcDeclareLanguageAlias { newzealand } { english }
2515 ⟨package⟩\zcDeclareLanguageAlias { UKenglish  } { english }
2516 ⟨package⟩\zcDeclareLanguageAlias { USenglish  } { english }
2517 ⟨*dict-english⟩
2518 namesep   = {\nobreakspace} ,
2519 pairsep   = {~and\nobreakspace} ,
2520 listsep   = {,~} ,
2521 lastsep   = {~and\nobreakspace} ,
2522 tpairsep  = {~and\nobreakspace} ,
2523 tlistsep  = {,~} ,
2524 tlastsep  = {,~and\nobreakspace} ,
2525 notesep   = {~} ,
2526 rangesep  = {~to\nobreakspace} ,
2527
2528 type = part ,
2529   Name-sg = Part ,
2530   name-sg = part ,
2531   Name-pl = Parts ,
2532   name-pl = parts ,
2533
2534 type = chapter ,
2535   Name-sg = Chapter ,
2536   name-sg = chapter ,
2537   Name-pl = Chapters ,
2538   name-pl = chapters ,
2539
2540 type = section ,
2541   Name-sg = Section ,
2542   name-sg = section ,
2543   Name-pl = Sections ,
2544   name-pl = sections ,
2545
2546 type = paragraph ,
2547   Name-sg = Paragraph ,
2548   name-sg = paragraph ,
2549   Name-pl = Paragraphs ,
2550   name-pl = paragraphs ,
```

```
2551    Name-sg-ab = Par. ,
2552    name-sg-ab = par. ,
2553    Name-pl-ab = Par. ,
2554    name-pl-ab = par. ,
2555
2556 type = appendix ,
2557    Name-sg = Appendix ,
2558    name-sg = appendix ,
2559    Name-pl = Appendices ,
2560    name-pl = appendices ,
2561
2562 type = page ,
2563    Name-sg = Page ,
2564    name-sg = page ,
2565    Name-pl = Pages ,
2566    name-pl = pages ,
2567    name-sg-ab = p. ,
2568    name-pl-ab = pp. ,
2569
2570 type = line ,
2571    Name-sg = Line ,
2572    name-sg = line ,
2573    Name-pl = Lines ,
2574    name-pl = lines ,
2575
2576 type = figure ,
2577    Name-sg = Figure ,
2578    name-sg = figure ,
2579    Name-pl = Figures ,
2580    name-pl = figures ,
2581    Name-sg-ab = Fig. ,
2582    name-sg-ab = fig. ,
2583    Name-pl-ab = Figs. ,
2584    name-pl-ab = figs. ,
2585
2586 type = table ,
2587    Name-sg = Table ,
2588    name-sg = table ,
2589    Name-pl = Tables ,
2590    name-pl = tables ,
2591
2592 type = item ,
2593    Name-sg = Item ,
2594    name-sg = item ,
2595    Name-pl = Items ,
2596    name-pl = items ,
2597
2598 type = footnote ,
2599    Name-sg = Footnote ,
2600    name-sg = footnote ,
2601    Name-pl = Footnotes ,
2602    name-pl = footnotes ,
2603
2604 type = note ,
```

69

```
2605    Name-sg = Note ,
2606    name-sg = note ,
2607    Name-pl = Notes ,
2608    name-pl = notes ,
2609
2610  type = equation ,
2611    Name-sg = Equation ,
2612    name-sg = equation ,
2613    Name-pl = Equations ,
2614    name-pl = equations ,
2615    Name-sg-ab = Eq. ,
2616    name-sg-ab = eq. ,
2617    Name-pl-ab = Eqs. ,
2618    name-pl-ab = eqs. ,
2619    refpre-in = {( ,
2620    refpos-in = )} ,
2621
2622  type = theorem ,
2623    Name-sg = Theorem ,
2624    name-sg = theorem ,
2625    Name-pl = Theorems ,
2626    name-pl = theorems ,
2627
2628  type = lemma ,
2629    Name-sg = Lemma ,
2630    name-sg = lemma ,
2631    Name-pl = Lemmas ,
2632    name-pl = lemmas ,
2633
2634  type = corollary ,
2635    Name-sg = Corollary ,
2636    name-sg = corollary ,
2637    Name-pl = Corollaries ,
2638    name-pl = corollaries ,
2639
2640  type = proposition ,
2641    Name-sg = Proposition ,
2642    name-sg = proposition ,
2643    Name-pl = Propositions ,
2644    name-pl = propositions ,
2645
2646  type = definition ,
2647    Name-sg = Definition ,
2648    name-sg = definition ,
2649    Name-pl = Definitions ,
2650    name-pl = definitions ,
2651
2652  type = proof ,
2653    Name-sg = Proof ,
2654    name-sg = proof ,
2655    Name-pl = Proofs ,
2656    name-pl = proofs ,
2657
2658  type = result ,
```

70

```
2659    Name-sg = Result ,
2660    name-sg = result ,
2661    Name-pl = Results ,
2662    name-pl = results ,
2663
2664 type = example ,
2665    Name-sg = Example ,
2666    name-sg = example ,
2667    Name-pl = Examples ,
2668    name-pl = examples ,
2669
2670 type = remark ,
2671    Name-sg = Remark ,
2672    name-sg = remark ,
2673    Name-pl = Remarks ,
2674    name-pl = remarks ,
2675
2676 type = algorithm ,
2677    Name-sg = Algorithm ,
2678    name-sg = algorithm ,
2679    Name-pl = Algorithms ,
2680    name-pl = algorithms ,
2681
2682 type = listing ,
2683    Name-sg = Listing ,
2684    name-sg = listing ,
2685    Name-pl = Listings ,
2686    name-pl = listings ,
2687
2688 type = exercise ,
2689    Name-sg = Exercise ,
2690    name-sg = exercise ,
2691    Name-pl = Exercises ,
2692    name-pl = exercises ,
2693
2694 type = solution ,
2695    Name-sg = Solution ,
2696    name-sg = solution ,
2697    Name-pl = Solutions ,
2698    name-pl = solutions ,
2699 ⟨/dict-english⟩
```

10.2   German

```
2700 ⟨package⟩\zcDeclareLanguage { german }
2701 ⟨package⟩\zcDeclareLanguageAlias { austrian     } { german }
2702 ⟨package⟩\zcDeclareLanguageAlias { germanb      } { german }
2703 ⟨package⟩\zcDeclareLanguageAlias { ngerman      } { german }
2704 ⟨package⟩\zcDeclareLanguageAlias { naustrian    } { german }
2705 ⟨package⟩\zcDeclareLanguageAlias { nswissgerman } { german }
2706 ⟨package⟩\zcDeclareLanguageAlias { swissgerman  } { german }
2707 ⟨*dict-german⟩

2708 namesep  = {\nobreakspace} ,
2709 pairsep  = {~und\nobreakspace} ,
```

```
2710 listsep  = {,~} ,
2711 lastsep  = {~und\nobreakspace} ,
2712 tpairsep = {~und\nobreakspace} ,
2713 tlistsep = {,~} ,
2714 tlastsep = {~und\nobreakspace} ,
2715 notesep  = {~} ,
2716 rangesep = {~bis\nobreakspace} ,
2717
2718 type = part ,
2719   Name-sg = Teil ,
2720   name-sg = Teil ,
2721   Name-pl = Teile ,
2722   name-pl = Teile ,
2723
2724 type = chapter ,
2725   Name-sg = Kapitel ,
2726   name-sg = Kapitel ,
2727   Name-pl = Kapitel ,
2728   name-pl = Kapitel ,
2729
2730 type = section ,
2731   Name-sg = Abschnitt ,
2732   name-sg = Abschnitt ,
2733   Name-pl = Abschnitte ,
2734   name-pl = Abschnitte ,
2735
2736 type = paragraph ,
2737   Name-sg = Absatz ,
2738   name-sg = Absatz ,
2739   Name-pl = Absätze ,
2740   name-pl = Absätze ,
2741
2742 type = appendix ,
2743   Name-sg = Anhang ,
2744   name-sg = Anhang ,
2745   Name-pl = Anhänge ,
2746   name-pl = Anhänge ,
2747
2748 type = page ,
2749   Name-sg = Seite ,
2750   name-sg = Seite ,
2751   Name-pl = Seiten ,
2752   name-pl = Seiten ,
2753
2754 type = line ,
2755   Name-sg = Zeile ,
2756   name-sg = Zeile ,
2757   Name-pl = Zeilen ,
2758   name-pl = Zeilen ,
2759
2760 type = figure ,
2761   Name-sg = Abbildung ,
2762   name-sg = Abbildung ,
2763   Name-pl = Abbildungen ,
```

```
2764    name-pl = Abbildungen ,
2765    Name-sg-ab = Abb. ,
2766    name-sg-ab = Abb. ,
2767    Name-pl-ab = Abb. ,
2768    name-pl-ab = Abb. ,
2769
2770  type = table ,
2771    Name-sg = Tabelle ,
2772    name-sg = Tabelle ,
2773    Name-pl = Tabellen ,
2774    name-pl = Tabellen ,
2775
2776  type = item ,
2777    Name-sg = Punkt ,
2778    name-sg = Punkt ,
2779    Name-pl = Punkte ,
2780    name-pl = Punkte ,
2781
2782  type = footnote ,
2783    Name-sg = Fußnote ,
2784    name-sg = Fußnote ,
2785    Name-pl = Fußnoten ,
2786    name-pl = Fußnoten ,
2787
2788  type = note ,
2789    Name-sg = Anmerkung ,
2790    name-sg = Anmerkung ,
2791    Name-pl = Anmerkungen ,
2792    name-pl = Anmerkungen ,
2793
2794  type = equation ,
2795    Name-sg = Gleichung ,
2796    name-sg = Gleichung ,
2797    Name-pl = Gleichungen ,
2798    name-pl = Gleichungen ,
2799    refpre-in = {(} ,
2800    refpos-in = {)} ,
2801
2802  type = theorem ,
2803    Name-sg = Theorem ,
2804    name-sg = Theorem ,
2805    Name-pl = Theoreme ,
2806    name-pl = Theoreme ,
2807
2808  type = lemma ,
2809    Name-sg = Lemma ,
2810    name-sg = Lemma ,
2811    Name-pl = Lemmata ,
2812    name-pl = Lemmata ,
2813
2814  type = corollary ,
2815    Name-sg = Korollar ,
2816    name-sg = Korollar ,
2817    Name-pl = Korollare ,
```

```
2818    name-pl = Korollare ,
2819
2820  type = proposition ,
2821    Name-sg = Satz ,
2822    name-sg = Satz ,
2823    Name-pl = Sätze ,
2824    name-pl = Sätze ,
2825
2826  type = definition ,
2827    Name-sg = Definition ,
2828    name-sg = Definition ,
2829    Name-pl = Definitionen ,
2830    name-pl = Definitionen ,
2831
2832  type = proof ,
2833    Name-sg = Beweis ,
2834    name-sg = Beweis ,
2835    Name-pl = Beweise ,
2836    name-pl = Beweise ,
2837
2838  type = result ,
2839    Name-sg = Ergebnis ,
2840    name-sg = Ergebnis ,
2841    Name-pl = Ergebnisse ,
2842    name-pl = Ergebnisse ,
2843
2844  type = example ,
2845    Name-sg = Beispiel ,
2846    name-sg = Beispiel ,
2847    Name-pl = Beispiele ,
2848    name-pl = Beispiele ,
2849
2850  type = remark ,
2851    Name-sg = Bemerkung ,
2852    name-sg = Bemerkung ,
2853    Name-pl = Bemerkungen ,
2854    name-pl = Bemerkungen ,
2855
2856  type = algorithm ,
2857    Name-sg = Algorithmus ,
2858    name-sg = Algorithmus ,
2859    Name-pl = Algorithmen ,
2860    name-pl = Algorithmen ,
2861
2862  type = listing ,
2863    Name-sg = Listing , % CHECK
2864    name-sg = Listing , % CHECK
2865    Name-pl = Listings , % CHECK
2866    name-pl = Listings , % CHECK
2867
2868  type = exercise ,
2869    Name-sg = Übungsaufgabe ,
2870    name-sg = Übungsaufgabe ,
2871    Name-pl = Übungsaufgaben ,
```

74

```
2872    name-pl = Übungsaufgaben ,
2873
2874  type = solution ,
2875    Name-sg = Lösung ,
2876    name-sg = Lösung ,
2877    Name-pl = Lösungen ,
2878    name-pl = Lösungen ,
2879  ⟨/dict-german⟩
```

10.3   French

```
2880  ⟨package⟩\zcDeclareLanguage { french }
2881  ⟨package⟩\zcDeclareLanguageAlias { acadian  } { french }
2882  ⟨package⟩\zcDeclareLanguageAlias { canadien } { french }
2883  ⟨package⟩\zcDeclareLanguageAlias { francais } { french }
2884  ⟨package⟩\zcDeclareLanguageAlias { frenchb  } { french }
2885  ⟨*dict-french⟩
2886  namesep  = {\nobreakspace} ,
2887  pairsep  = {~et\nobreakspace} ,
2888  listsep  = {,~} ,
2889  lastsep  = {~et\nobreakspace} ,
2890  tpairsep = {~et\nobreakspace} ,
2891  tlistsep = {,~} ,
2892  tlastsep = {~et\nobreakspace} ,
2893  notesep  = {~} ,
2894  rangesep = {~à\nobreakspace} ,
2895
2896  type = part ,
2897    Name-sg = Partie ,
2898    name-sg = partie ,
2899    Name-pl = Parties ,
2900    name-pl = parties ,
2901
2902  type = chapter ,
2903    Name-sg = Chapitre ,
2904    name-sg = chapitre ,
2905    Name-pl = Chapitres ,
2906    name-pl = chapitres ,
2907
2908  type = section ,
2909    Name-sg = Section ,
2910    name-sg = section ,
2911    Name-pl = Sections ,
2912    name-pl = sections ,
2913
2914  type = paragraph ,
2915    Name-sg = Paragraphe ,
2916    name-sg = paragraphe ,
2917    Name-pl = Paragraphes ,
2918    name-pl = paragraphes ,
2919
2920  type = appendix ,
2921    Name-sg = Annexe ,
2922    name-sg = annexe ,
```

```
2923    Name-pl = Annexes ,
2924    name-pl = annexes ,
2925
2926  type = page ,
2927    Name-sg = Page ,
2928    name-sg = page ,
2929    Name-pl = Pages ,
2930    name-pl = pages ,
2931
2932  type = line ,
2933    Name-sg = Ligne ,
2934    name-sg = ligne ,
2935    Name-pl = Lignes ,
2936    name-pl = lignes ,
2937
2938  type = figure ,
2939    Name-sg = Figure ,
2940    name-sg = figure ,
2941    Name-pl = Figures ,
2942    name-pl = figures ,
2943
2944  type = table ,
2945    Name-sg = Table ,
2946    name-sg = table ,
2947    Name-pl = Tables ,
2948    name-pl = tables ,
2949
2950  type = item ,
2951    Name-sg = Point ,
2952    name-sg = point ,
2953    Name-pl = Points ,
2954    name-pl = points ,
2955
2956  type = footnote ,
2957    Name-sg = Note ,
2958    name-sg = note ,
2959    Name-pl = Notes ,
2960    name-pl = notes ,
2961
2962  type = note ,
2963    Name-sg = Note ,
2964    name-sg = note ,
2965    Name-pl = Notes ,
2966    name-pl = notes ,
2967
2968  type = equation ,
2969    Name-sg = Équation ,
2970    name-sg = équation ,
2971    Name-pl = Équations ,
2972    name-pl = équations ,
2973    refpre-in = {() ,
2974    refpos-in = {)} ,
2975
2976  type = theorem ,
```

```
2977    Name-sg = Théorème ,
2978    name-sg = théorème ,
2979    Name-pl = Théorèmes ,
2980    name-pl = théorèmes ,
2981
2982 type = lemma ,
2983    Name-sg = Lemme ,
2984    name-sg = lemme ,
2985    Name-pl = Lemmes ,
2986    name-pl = lemmes ,
2987
2988 type = corollary ,
2989    Name-sg = Corollaire ,
2990    name-sg = corollaire ,
2991    Name-pl = Corollaires ,
2992    name-pl = corollaires ,
2993
2994 type = proposition ,
2995    Name-sg = Proposition ,
2996    name-sg = proposition ,
2997    Name-pl = Propositions ,
2998    name-pl = propositions ,
2999
3000 type = definition ,
3001    Name-sg = Définition ,
3002    name-sg = définition ,
3003    Name-pl = Définitions ,
3004    name-pl = définitions ,
3005
3006 type = proof ,
3007    Name-sg = Démonstration ,
3008    name-sg = démonstration ,
3009    Name-pl = Démonstrations ,
3010    name-pl = démonstrations ,
3011
3012 type = result ,
3013    Name-sg = Résultat ,
3014    name-sg = résultat ,
3015    Name-pl = Résultats ,
3016    name-pl = résultats ,
3017
3018 type = example ,
3019    Name-sg = Exemple ,
3020    name-sg = exemple ,
3021    Name-pl = Exemples ,
3022    name-pl = exemples ,
3023
3024 type = remark ,
3025    Name-sg = Remarque ,
3026    name-sg = remarque ,
3027    Name-pl = Remarques ,
3028    name-pl = remarques ,
3029
3030 type = algorithm ,
```

```
3031    Name-sg = Algorithme ,
3032    name-sg = algorithme ,
3033    Name-pl = Algorithmes ,
3034    name-pl = algorithmes ,
3035
3036 type = listing ,
3037    Name-sg = Liste ,
3038    name-sg = liste ,
3039    Name-pl = Listes ,
3040    name-pl = listes ,
3041
3042 type = exercise ,
3043    Name-sg = Exercice ,
3044    name-sg = exercice ,
3045    Name-pl = Exercices ,
3046    name-pl = exercices ,
3047
3048 type = solution ,
3049    Name-sg = Solution ,
3050    name-sg = solution ,
3051    Name-pl = Solutions ,
3052    name-pl = solutions ,
3053 ⟨/dict-french⟩
```

10.4 Portuguese

```
3054 ⟨package⟩\zcDeclareLanguage { portuguese }
3055 ⟨package⟩\zcDeclareLanguageAlias { brazilian } { portuguese }
3056 ⟨package⟩\zcDeclareLanguageAlias { brazil   } { portuguese }
3057 ⟨package⟩\zcDeclareLanguageAlias { portuges } { portuguese }
3058 ⟨*dict-portuguese⟩
3059 namesep  = {\nobreakspace} ,
3060 pairsep  = {~e\nobreakspace} ,
3061 listsep  = {,~} ,
3062 lastsep  = {~e\nobreakspace} ,
3063 tpairsep = {~e\nobreakspace} ,
3064 tlistsep = {,~} ,
3065 tlastsep = {~e\nobreakspace} ,
3066 notesep  = {~} ,
3067 rangesep = {~a\nobreakspace} ,
3068
3069 type = part ,
3070    Name-sg = Parte ,
3071    name-sg = parte ,
3072    Name-pl = Partes ,
3073    name-pl = partes ,
3074
3075 type = chapter ,
3076    Name-sg = Capítulo ,
3077    name-sg = capítulo ,
3078    Name-pl = Capítulos ,
3079    name-pl = capítulos ,
3080
3081 type = section ,
```

```
3082    Name-sg = Seção ,
3083    name-sg = seção ,
3084    Name-pl = Seções ,
3085    name-pl = seções ,
3086
3087 type = paragraph ,
3088    Name-sg = Parágrafo ,
3089    name-sg = parágrafo ,
3090    Name-pl = Parágrafos ,
3091    name-pl = parágrafos ,
3092    Name-sg-ab = Par. ,
3093    name-sg-ab = par. ,
3094    Name-pl-ab = Par. ,
3095    name-pl-ab = par. ,
3096
3097 type = appendix ,
3098    Name-sg = Apêndice ,
3099    name-sg = apêndice ,
3100    Name-pl = Apêndices ,
3101    name-pl = apêndices ,
3102
3103 type = page ,
3104    Name-sg = Página ,
3105    name-sg = página ,
3106    Name-pl = Páginas ,
3107    name-pl = páginas ,
3108    name-sg-ab = p. ,
3109    name-pl-ab = pp. ,
3110
3111 type = line ,
3112    Name-sg = Linha ,
3113    name-sg = linha ,
3114    Name-pl = Linhas ,
3115    name-pl = linhas ,
3116
3117 type = figure ,
3118    Name-sg = Figura ,
3119    name-sg = figura ,
3120    Name-pl = Figuras ,
3121    name-pl = figuras ,
3122    Name-sg-ab = Fig. ,
3123    name-sg-ab = fig. ,
3124    Name-pl-ab = Figs. ,
3125    name-pl-ab = figs. ,
3126
3127 type = table ,
3128    Name-sg = Tabela ,
3129    name-sg = tabela ,
3130    Name-pl = Tabelas ,
3131    name-pl = tabelas ,
3132
3133 type = item ,
3134    Name-sg = Item ,
3135    name-sg = item ,
```

```
3136    Name-pl = Itens ,
3137    name-pl = itens ,
3138
3139 type = footnote ,
3140    Name-sg = Nota ,
3141    name-sg = nota ,
3142    Name-pl = Notas ,
3143    name-pl = notas ,
3144
3145 type = note ,
3146    Name-sg = Nota ,
3147    name-sg = nota ,
3148    Name-pl = Notas ,
3149    name-pl = notas ,
3150
3151 type = equation ,
3152    Name-sg = Equação ,
3153    name-sg = equação ,
3154    Name-pl = Equações ,
3155    name-pl = equações ,
3156    Name-sg-ab = Eq. ,
3157    name-sg-ab = eq. ,
3158    Name-pl-ab = Eqs. ,
3159    name-pl-ab = eqs. ,
3160    refpre-in = {(} ,
3161    refpos-in = {)} ,
3162
3163 type = theorem ,
3164    Name-sg = Teorema ,
3165    name-sg = teorema ,
3166    Name-pl = Teoremas ,
3167    name-pl = teoremas ,
3168
3169 type = lemma ,
3170    Name-sg = Lema ,
3171    name-sg = lema ,
3172    Name-pl = Lemas ,
3173    name-pl = lemas ,
3174
3175 type = corollary ,
3176    Name-sg = Corolário ,
3177    name-sg = corolário ,
3178    Name-pl = Corolários ,
3179    name-pl = corolários ,
3180
3181 type = proposition ,
3182    Name-sg = Proposição ,
3183    name-sg = proposição ,
3184    Name-pl = Proposições ,
3185    name-pl = proposições ,
3186
3187 type = definition ,
3188    Name-sg = Definição ,
3189    name-sg = definição ,
```

```
3190    Name-pl = Definições ,
3191    name-pl = definições ,
3192
3193 type = proof ,
3194    Name-sg = Demonstração ,
3195    name-sg = demonstração ,
3196    Name-pl = Demonstrações ,
3197    name-pl = demonstrações ,
3198
3199 type = result ,
3200    Name-sg = Resultado ,
3201    name-sg = resultado ,
3202    Name-pl = Resultados ,
3203    name-pl = resultados ,
3204
3205 type = example ,
3206    Name-sg = Exemplo ,
3207    name-sg = exemplo ,
3208    Name-pl = Exemplos ,
3209    name-pl = exemplos ,
3210
3211 type = remark ,
3212    Name-sg = Observação ,
3213    name-sg = observação ,
3214    Name-pl = Observações ,
3215    name-pl = observações ,
3216
3217 type = algorithm ,
3218    Name-sg = Algoritmo ,
3219    name-sg = algoritmo ,
3220    Name-pl = Algoritmos ,
3221    name-pl = algoritmos ,
3222
3223 type = listing ,
3224    Name-sg = Listagem ,
3225    name-sg = listagem ,
3226    Name-pl = Listagens ,
3227    name-pl = listagens ,
3228
3229 type = exercise ,
3230    Name-sg = Exercício ,
3231    name-sg = exercício ,
3232    Name-pl = Exercícios ,
3233    name-pl = exercícios ,
3234
3235 type = solution ,
3236    Name-sg = Solução ,
3237    name-sg = solução ,
3238    Name-pl = Soluções ,
3239    name-pl = soluções ,
3240 ⟨/dict-portuguese⟩
```

10.5   Spanish

```
3241 ⟨package⟩\zcDeclareLanguage { spanish }
3242 ⟨*dict-spanish⟩

3243 namesep  = {\nobreakspace} ,
3244 pairsep  = {~y\nobreakspace} ,
3245 listsep  = {,~} ,
3246 lastsep  = {~y\nobreakspace} ,
3247 tpairsep = {~y\nobreakspace} ,
3248 tlistsep = {,~} ,
3249 tlastsep = {~y\nobreakspace} ,
3250 notesep  = {~} ,
3251 rangesep = {~a\nobreakspace} ,
3252
3253 type = part ,
3254   Name-sg = Parte ,
3255   name-sg = parte ,
3256   Name-pl = Partes ,
3257   name-pl = partes ,
3258
3259 type = chapter ,
3260   Name-sg = Capítulo ,
3261   name-sg = capítulo ,
3262   Name-pl = Capítulos ,
3263   name-pl = capítulos ,
3264
3265 type = section ,
3266   Name-sg = Sección ,
3267   name-sg = sección ,
3268   Name-pl = Secciones ,
3269   name-pl = secciones ,
3270
3271 type = paragraph ,
3272   Name-sg = Párrafo ,
3273   name-sg = párrafo ,
3274   Name-pl = Párrafos ,
3275   name-pl = párrafos ,
3276
3277 type = appendix ,
3278   Name-sg = Apéndice ,
3279   name-sg = apéndice ,
3280   Name-pl = Apéndices ,
3281   name-pl = apéndices ,
3282
3283 type = page ,
3284   Name-sg = Página ,
3285   name-sg = página ,
3286   Name-pl = Páginas ,
3287   name-pl = páginas ,
3288
3289 type = line ,
3290   Name-sg = Línea ,
3291   name-sg = línea ,
3292   Name-pl = Líneas ,
3293   name-pl = líneas ,
```

```
3294
3295 type = figure ,
3296   Name-sg = Figura ,
3297   name-sg = figura ,
3298   Name-pl = Figuras ,
3299   name-pl = figuras ,
3300
3301 type = table ,
3302   Name-sg = Cuadro ,
3303   name-sg = cuadro ,
3304   Name-pl = Cuadros ,
3305   name-pl = cuadros ,
3306
3307 type = item ,
3308   Name-sg = Punto ,
3309   name-sg = punto ,
3310   Name-pl = Puntos ,
3311   name-pl = puntos ,
3312
3313 type = footnote ,
3314   Name-sg = Nota ,
3315   name-sg = nota ,
3316   Name-pl = Notas ,
3317   name-pl = notas ,
3318
3319 type = note ,
3320   Name-sg = Nota ,
3321   name-sg = nota ,
3322   Name-pl = Notas ,
3323   name-pl = notas ,
3324
3325 type = equation ,
3326   Name-sg = Ecuación ,
3327   name-sg = ecuación ,
3328   Name-pl = Ecuaciones ,
3329   name-pl = ecuaciones ,
3330   refpre-in = {(} ,
3331   refpos-in = {)} ,
3332
3333 type = theorem ,
3334   Name-sg = Teorema ,
3335   name-sg = teorema ,
3336   Name-pl = Teoremas ,
3337   name-pl = teoremas ,
3338
3339 type = lemma ,
3340   Name-sg = Lema ,
3341   name-sg = lema ,
3342   Name-pl = Lemas ,
3343   name-pl = lemas ,
3344
3345 type = corollary ,
3346   Name-sg = Corolario ,
3347   name-sg = corolario ,
```

```
3348    Name-pl = Corolarios ,
3349    name-pl = corolarios ,
3350
3351 type = proposition ,
3352    Name-sg = Proposición ,
3353    name-sg = proposición ,
3354    Name-pl = Proposiciones ,
3355    name-pl = proposiciones ,
3356
3357 type = definition ,
3358    Name-sg = Definición ,
3359    name-sg = definición ,
3360    Name-pl = Definiciones ,
3361    name-pl = definiciones ,
3362
3363 type = proof ,
3364    Name-sg = Demostración ,
3365    name-sg = demostración ,
3366    Name-pl = Demostraciones ,
3367    name-pl = demostraciones ,
3368
3369 type = result ,
3370    Name-sg = Resultado ,
3371    name-sg = resultado ,
3372    Name-pl = Resultados ,
3373    name-pl = resultados ,
3374
3375 type = example ,
3376    Name-sg = Ejemplo ,
3377    name-sg = ejemplo ,
3378    Name-pl = Ejemplos ,
3379    name-pl = ejemplos ,
3380
3381 type = remark ,
3382    Name-sg = Observación ,
3383    name-sg = observación ,
3384    Name-pl = Observaciones ,
3385    name-pl = observaciones ,
3386
3387 type = algorithm ,
3388    Name-sg = Algoritmo ,
3389    name-sg = algoritmo ,
3390    Name-pl = Algoritmos ,
3391    name-pl = algoritmos ,
3392
3393 type = listing ,
3394    Name-sg = Listado ,
3395    name-sg = listado ,
3396    Name-pl = Listados ,
3397    name-pl = listados ,
3398
3399 type = exercise ,
3400    Name-sg = Ejercicio ,
3401    name-sg = ejercicio ,
```

```
3402    Name-pl = Ejercicios ,
3403    name-pl = ejercicios ,
3404
3405 type = solution ,
3406    Name-sg = Solución ,
3407    name-sg = solución ,
3408    Name-pl = Soluciones ,
3409    name-pl = soluciones ,
3410 ⟨/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.

86

91