

The latex-lab-enumitem package

Emulating enumitem

L^AT_EX Project*

v0.80g 2026-06-28

Abstract

The following code implements an emulation of `enumitem` usable with the Tagging Support Code. It does *not* emulate every key and command of `enumitem`. Also syntax and behaviour can differ in place.

1 Introduction

The `enumitem` package offers customizable and enhanced list environments but is not compatible with the Tagging Support Code where lists are reimplemented with templates.

The following code partly emulates `enumitem` and should be loaded instead of the package.

2 Key emulation

A large part of the `enumitem` functionality is provided through keys. Such keys can be set in the optional argument of single lists and globally for some or all lists in the `\setlist` command. Some keys are simple interfaces to list parameters, other have quite complicated effects, e.g. if they force recalculations of dependent parameters.

With the new L^AT_EX implementation lists do have an optional argument too which process a key-value list. The key names differ to the one of `enumitem` but in a number of cases a simple mapping is possible. The L^AT_EX key names are noted in the following tabular in the second column. They can be used instead of the `enumitem` keys (and will be a bit faster). The third column shows if the `enumitem` key has been already emulated and is usable in the optional argument. The fourth column shows if it is usable in `\setlist`.

Table 1: List of `enumitem` keys

enumitem key	L ^A T _E X key	opt. arg.	<code>\setlist</code>	comment
<code>label</code>	(related: <code>item-label</code>)			see key description
<code>label*</code>				
<code>ref</code>				
<code>font,format</code>	<code>label-format</code>	yes		see key description!!
<code>format</code>				

*Initial implementation done by Ulrike Fischer

Table 1: List of enumitem keys

enumitem key	L ^A T _E X key	opt. arg.	\setlist	comment
align	label-align	yes		see key description
topsep	begin-vspace + end-vspace	yes		
partopsep	begin-extra-vspace + end-extra-vspace	yes		
parsep	para-vspace	yes		
itemsep	item-vspace	yes		
labelindent				
labelindent*				
leftmargin	left-margin			see key description
rightmargin	right-margin			see key description
listparindent	para-indent			see key description
itemindent	item-indent			see key description
labelsep	label-sep			see key description
labelsep*				
labelwidth	label-width			see key description
left				
widest				
widest*				
start	start	yes	?	see key description
resume	resume	yes	?	
resume*				
series				
beginpenalty	begin-penalty	yes	?	
midpenalty	item-penalty	yes	?	
endpenalty	end-penalty	yes	?	
before				
before*				
after				
after*				
first				
first*				
style				
noitemsep	<i>meta-key</i>	yes		
nosep	<i>meta-key</i>	yes		
wide		partly		
itemjoin				
itemjoin*				
afterlabel				
mode				

3 Package options

enumitem has a number of package options, but none of them will be supported by the emulation: adding support for the `enumerate` syntax (`shortlabels`) is not planned; inline lists are not yet implemented but once they are they will be available always.

Table 2: List of enumitem package options

enumitem option
shortlabels
inline
ignoredisplayed
series=override
sizes
loadonly

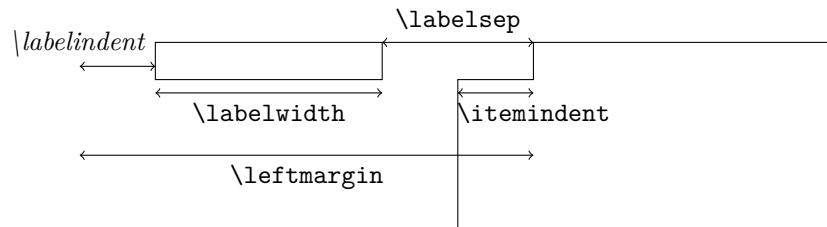
4 Commands

Table 3: List of enumitem user commands

name	emulated	comment
<code>\SetLabelAlign</code>		
<code>\DrawEnumitemLabel</code>		
<code>\labelindent</code>		
<code>\EnumitemId</code>		
<code>\SetEnumitemKey</code>	yes	
<code>\SetEnumitemValue</code>		
<code>\SetEnumerateShortLabel</code>		
<code>\newlist</code>	yes	
<code>\renewlist</code>		
<code>\setlist</code>	yes	no size-dependent settings
<code>\setlistdepth</code>		
<code>\AddEnumerateCounter</code>		
<code>\restartlist</code>		
<code>\SetEnumitemSize</code>		

5 Calculating values

When setting up the dimension of lists the values related to the left margin and placement of the label are typically the most challenging as four dimensions are involved (which can be negative). `enumitem` introduces a fifth dimension `\labelindent`.



The relation between the five values is set through the following equation:

$$\text{\labelindent} + \text{\labelwidth} + \text{\labelsep} = \text{\leftmargin} + \text{\itemindent}$$

So obviously one of the dimensions is redundant and can be calculated if the others are given. By default `enumitem` calculates the new, “fake” dimension `\labelindent`

but it is possible to give `\labelindent` a specific value and declare that another one is calculated by using `!` as value.

Calculation should happen after all keys are set. The code here therefore executes a key `calculate` in the list code which looks which key is currently the dependent one and calculates its value.

`enumitem` also offers an option to calculate `\labelwidth` based on the widest entry (which can be declared with the `widest` key) by using a star `*` as value.

6 Alignment

```
1 <*package>
```

7 Implementation

```
2 \ProvidesExplPackage {latex-lab-enumitem} {\ltlabenumdate} {\ltlabenumversion}
3   {Emulating enumitem}
```

7.1 Key emulation

7.1.1 label

check all this code

The key `item-label` from the new L^AT_EX list code changes the label representation from e.g. `\labelenumi` to the key value. Internally the representation of the current list is stored in `\l_block_item_label_tl`. It does not change `\labelenumi`, `\theenumi` or `\p@enumi` and so also doesn't affect references.

The key `label` from `enumitem` works quite differently: it changes the label representation command `\labelenumi`, the counter representation `\theenumi` and sets `\p@enumi` to empty. So by default the reference is identical to the label representation and if a different reference is wanted it must be set with the `ref` key. To allow to use `\labelenumi` in nested list to create chained labels `enumitem` replaces all `\roman*` by `\roman{enumi}` (and similar for the other counter representations). The replacement code uses expansion and so `enumitem` will error if the star is used with an unknown counter representation like `label=\fnsymbol*` or `\alphalph`. Such unknown counter representation must first be added with `\AddEnumerateCounter`.

The new code is less fragile. Counter representation must only be declared with `\AddEnumerateCounter` if the star-counter is used outside the current list (e.g. with the `label*` key).

```
4 <@@=block> % we might want a different module in the end
```

At first a rather crude (slow) method to replace e.g. `\roman*` by `\roman{enumi}` in the label representation. TODO: speed up.

```
5 \clist_new:N \l_block_normalize_cnt_clist
6 \clist_set:Nn \l_block_normalize_cnt_clist
7   {\alph,\Alph,\roman,\Roman,\arabic,\fnsymbol}
```

This command should only be used if `\l_block_counter_tl` is not empty!

```
8 \cs_new:Npn \__block_normalize_label:N #1
9 {
10   \clist_map_inline:Nn \l_block_normalize_cnt_clist
```

```

11     {
12       \tl_replace_all:Nne#1 {##1*}{\exp_not:N##1{\l__block_counter_tl}}
13     }
14   }
15   \cs_generate_variant:Nn\__block_normalize_label:N{c}

16   \keys_define:nn{template/list/std}
17   {
18     label .code:n =
19     {
20       \tl_if_empty:NTF\l__block_counter_tl
21       {
22         \tl_if_eq:NnTF\l__block_inner_instance_tl{itemize}
23         {
24           \tl_set:cn{labelitem\int_to_roman:n{\l__block_inner_level_counter_tl}}{#1}
25         }
26         {
27           \tl_set:cn{label\l__block_inner_instance_tl\int_to_roman:n{\l__block_inner_level_counter_tl}}{#1}
28         }
29       }
30       {
31         \tl_set:cn{label\l__block_counter_tl}{#1}
32         \__block_normalize_label:c{label\l__block_counter_tl}
33         \tl_set_eq:cc{the\l__block_counter_tl}{label\l__block_counter_tl}
34         \tl_set:cn{p@\l__block_counter_tl}{#1}
35       }
36     }
37   }

```

7.1.2 label*

TODO:

7.1.3 ref

TODO:

7.1.4 font, format

```

38   \keys_define:nn{template/item/std}
39   { font .meta:n = {label-format={#1{##1}}}}
40   \keys_define:nn{template/item/std}
41   { format .meta:n = {label-format={#1{##1}}}}

```

7.1.5 align

Note: this also supports the value center but parleft is not implemented yet. TODO: test for differences in behavior.

```

42   \keys_define:nn{template/list/std}
43   { align .meta:n = {label-align=#1}}

```

7.1.6 topsep

```
44 \keys_define:nn{template/block/std}  
45   { topsep .meta:n = { begin-vspace=#1,  
46                       end-vspace=#1 }}
```

7.1.7 partopsep

```
47 \keys_define:nn{template/block/std}  
48   { partopsep .meta:n = { begin-extra-vspace=#1,  
49                           end-extra-vspace=#1 }}
```

7.1.8 parsep

```
50 \keys_define:nn{template/block/std}  
51   { parsep .meta:n = {para-vspace=#1}}
```

7.1.9 itemsep

```
52 \keys_define:nn{template/block/std}  
53   { itemsep .meta:n = {item-vspace=#1}}
```

7.1.10 leftmargin

TODO: handle special values ! and *.

Special values do not work with L^AT_EX key at the moment as it is a register!

```
54 \keys_define:nn{template/block/std}  
55   { leftmargin .meta:n = {left-margin=#1}}
```

7.1.11 rightmargin

TODO: handle special values ! and *.

Special values do not work with L^AT_EX key!

```
56 \keys_define:nn{template/block/std}  
57   { rightmargin .meta:n = {right-margin=#1}}
```

7.1.12 listparindent

TODO: handle special values ! and *.

Special values do not work with L^AT_EX key!

```
58 \keys_define:nn{template/block/std}  
59   { listparindent .meta:n = {para-indent=#1}}
```

7.1.13 itemindent

TODO: handle special values ! and *.

Special values do not work with L^AT_EX key!

```
60 \keys_define:nn{template/list/std}  
61   { itemindent .meta:n = {item-indent=#1}}
```

7.1.14 `labelsep`, `labelsep*`

TODO: handle special values ! and *.
Special values do not work with $\LaTeX$ key!

```
62 \keys_define:nn{template/list/std}  
63   { labelsep .meta:n = {label-sep=#1}}
```

7.1.15 `labelwidth`

TODO: handle special values ! and *.
Special values do not work with $\LaTeX$ key!

```
64 \keys_define:nn{template/list/std}  
65   { labelwidth .meta:n = {label-width=#1}}
```

7.1.16 `labelindent`, `labelindent*`

TODO:, see also `\labelindent`, note special value ! and *

7.1.17 `left`

TODO:

7.1.18 `widest`, `widest*`

TODO:

7.1.19 `start`

TODO: Test with `setlist`

7.1.20 `resume`

TODO: Test with `setlist`.

The behavior of the key is different to `enumitem`, where grouping of the environments matters: in `enumitem` an `enumerate` that is e.g. in `quote` environment can not be resumed outside of the `quote`. With the $\LaTeX$ code grouping does not matter.

7.1.21 `resume*`

TODO: decide if it should be implemented

7.1.22 `series`

TODO: decide if it should be implemented

7.1.23 `beginpenalty`, `midpenalty`, `endpenalty`

```
66 \keys_define:nn{template/block/std}  
67   {  
68     beginpenalty .meta:n = {begin-penalty=#1},  
69     endpenalty .meta:n   = {end-penalty=#1},  
70     midpenalty .meta:n   = {item-penalty=#1}  
71   }
```

7.1.24 before, before*

TODO: describe behavior (second argument of list does not make sense) Implement with hooks?

7.1.25 after, after*

TODO: implement with hooks?

7.1.26 first, first*

TODO: implement with hooks?

7.1.27 style

TODO: values for description lists: standard, unboxed, nextline, sameline, multiline

7.1.28 noitemsep, nosep

```
72 \keys_define:nn{template/blockenv/std}  
73 {  
74   nosep .meta:n =  
75   {  
76     begin-vspace=0pt,  
77     begin-extra-vspace=0pt,  
78     end-vspace=0pt,  
79     end-extra-vspace=0pt,  
80     item-vspace=0pt,  
81     para-vspace=0pt  
82   }  
83 }  
84 \keys_define:nn{template/blockenv/std}  
85 {  
86   noitemsep .meta:n =  
87   {  
88     item-vspace=0pt,  
89     para-vspace=0pt  
90   }  
91 }  
92
```

7.1.29 wide

TODO: calculated value should be delayed ...

```
93 \keys_define:nn{template/blockenv/std}  
94 {  
95   wide .meta:n =  
96   {  
97     label-align=left,  
98     para-indent=#1,  
99     left-margin=0pt,  
100    label-width=0pt,  
101    item-indent=\dimeval{#1+\labelsep} %should be delayed ....  
102  },
```

```

103   wide .default:n = \parindent
104 }

```

7.1.30 `itemjoin`, `itemjoin*`, `afterlabel`

TODO

7.1.31 `mode`

TODO

7.2 Package options

7.2.1 `shortlabels`

7.2.2 `inline`

TODO, creates three environments `enumerate*`, `itemize*` and `description*`

7.2.3 `sizes`

7.2.4 `loadonly`

7.3 Command emulation

7.3.1 `\SetLabelAlign`

7.3.2 `\DrawEnumitemLabel`

7.3.3 `\labelindent`

see also key `labelindent`

7.3.4 `\EnumitemId`

7.3.5 `\SetEnumitemKey`

The `\SetEnumitemKey` defines shortcuts. We can defines them as `.meta:n` on the `block` level. If they are for the inner instances, they get passed down as necessary (this is slightly less efficient than defining them at the right level, but makes life easier).

```

105 \NewDocumentCommand \SetEnumitemKey {mm} {
106   \keys_define:nn { template/block/std }
107     { #1 .meta:n = { #2 } }
108 }

```

As an example, something like `noitemsep` could have been defined as

```
\SetEnumitemKey{noitemsep}{ itemsep=0pt, parsep=0pt }
```

And this can even be done recursively, e.g.,

```
\SetEnumitemKey{nosep} { noitemsep, topsep=0pt, partopsep=0pt }
```

7.3.6 \SetEnumitemValue

7.3.7 \SetEnumerateShortLabel

7.3.8 \newlist

The `\newlist` command allows to define new lists which clones the standard lists. The last number describes the maximum number of levels. Note that with `itemize` and `description` at most 6 levels are allowed. This could be changed with

```
\setcounter{maxblocklevels}{7}
\DeclareInstance{block}{std-list-7}{display}{}
```

but as `enumitem` doesn't allow more levels either the code does not force it.

Below is another implementation that supports arbitrarily many levels. At some point we need to decide what we want to do here and unify the code. Right now both sample implementations have their restrictions.

`\newlist`

```
109 \NewDocumentCommand\newlist{mmm} %name, type, number
110 {
111   \str_case:nnF{#2}
112   {
113     {itemize}      {\__block_setup_itemize:nn{#1}{#3}}
114     {enumerate}    {\__block_setup_enumerate:nn{#1}{#3}}
115     {description} {\__block_setup_description:nn{#1}{#3}}
116   }

```

TODO: message

```
117   {\typeout{unknown-list~type~#2}
118 }

```

TODO: For supporting `\setlist` we need to set up `\enitdp@#1` if we use this implementation.

```
119 }
120 %

```

(End of definition for \newlist. This function is documented on page ??.)

`\__block_setup_itemize:nn`

```
121 \cs_new_protected:Npn \__block_setup_itemize:nn #1#2 %#1 name of new list, #2 max levels
122 {
123   \DeclareInstanceCopy{blockenv}{#1}{itemize}
124   \EditInstance{blockenv}{#1}{inner-instance=#1}
125   \NewDocumentEnvironment{#1}{0}{}
126   { \SimpleBlockEnv {#1} {max-inner-levels= \int_min:nn{\c@maxblocklevels}{#2},##1} }
127   { \BlockEnvEnd }
128   \int_step_inline:nnn{1}{\int_min:nn{\c@maxblocklevels}{#2}}
129   {
130     \IfInstanceExistsTF{list}{itemize-##1}
131     {
132       \DeclareInstanceCopy{list}{#1-##1}{itemize-##1}
133     }

```

The default label for lists below 4 is simply `\labelitemi`.

```

134     {
135         \ExpandArgs{c}
136         \providecommand{labelitem\int_to_roman:n{##1}}{\labelitemi}
137         \DeclareInstance{list}{#1-##1}{std}
138         { item-label = \use:c{labelitem\int_to_roman:n{##1}}}
139     }
140 }
141 }

```

(End of definition for `\__block_setup_itemize:nn`.)

`\__block_setup_description:nn`

```

142 \cs_new_protected:Npn \__block_setup_description:nn #1#2
143 {
144     \DeclareInstanceCopy{blockenv}{#1}{description}
145     \EditInstance{blockenv}{#1}{inner-instance=#1}
146     \DeclareInstanceCopy{list}{#1}{description}
147     \NewDocumentEnvironment{#1}{0{}}
148     { \SimpleBlockEnv{#1} {max-inner-levels= \int_min:nn{\c@maxblocklevels}{#2},##1} }
149     { \BlockEnvEnd }
150 }

```

(End of definition for `\__block_setup_description:nn`.)

`\__block_setup_enumerate:nn`

```

151 \cs_new_protected:Npn \__block_setup_enumerate:nn #1#2
152 {
153     \DeclareInstanceCopy{blockenv}{#1}{enumerate}
154     \EditInstance{blockenv}{#1}{inner-instance=#1}
155     \NewDocumentEnvironment{#1}{0{}}
156     { \SimpleBlockEnv {#1} {max-inner-levels= #2,##1} }
157     { \BlockEnvEnd }
158     \int_step_inline:nnn{1}{#2}
159     {
160         \newcounter{#1\int_to_roman:n{##1}}
161         \ExpandArgs{c}
162         \newcommand{label#1\int_to_roman:n{##1}}
163         {\arabic{#1\int_to_roman:n{##1}}.}
164         \DeclareInstance{list}{#1-##1}{std}
165         {
166             item-label = \use:c{label#1\int_to_roman:n{##1}} ,
167             counter    = {#1\int_to_roman:n{##1}}
168         }
169     }
170 }

```

(End of definition for `\__block_setup_enumerate:nn`.)

`\newlist` This is an alternative implementation (not necessarily better).

```

171 \cs_set_protected:Npn \newlist #1#2#3 {
172     \DeclareDocumentEnvironment {#1} { !0{ } }
173     { \SimpleBlockEnv {#1}{##1} } { \BlockEnvEnd }
174 %
175     \DeclareInstanceCopy {blockenv} {#1} {#2}

```

```

176 %
177 % If #3 > maxblocklevels add additional std-list instances as necessary
178 %
179 \int_compare:nNtT { \value{maxblocklevels} } < {#3}
180 {
181   \int_step_inline:nnn { \value{maxblocklevels} + 1 } {#3}
182   {
183     \DeclareInstanceCopy {block} {std-list-##1} {std-list-\int_eval:n{ ##1 - 1 }}
184 % not sure if we should set up those as well ...
185     \DeclareInstanceCopy {block} {std-display-##1} {std-display-1}
186     \DeclareInstanceCopy {block} {quote-##1} {quote-1}
187     \DeclareInstanceCopy {block} {quotation-##1} {quotation-1}
188     \DeclareInstanceCopy {block} {verbatim-##1} {verbatim-1}
189   }
190   \setcounter{maxblocklevels}{#3}
191 }
192 %
193 \IfInstanceExistsTF{list}{#2} % no levels (as with description)
194 {
195   \EditInstance {blockenv} {#1} {
196     name = #1
197     ,inner-instance = #1
198   }
199 %
200   \DeclareInstanceCopy {list} {#1} {#2}
201 }
202 {
203   \int_new:c { g__block_ #1 _depth_int }
204 %
205   \EditInstance {blockenv} {#1} {
206     name = #1
207     ,inner-instance = #1
208     ,max-inner-levels = #3
209     ,inner-level-counter:c = g__block_ #1 _depth_int
210   }
211 %
212   \int_step_inline:nn {#3}
213   { \__block_setup_enum_list_instance:nnn {#1}{#2}{##1} }
214 }
215 %

```

For supporting \setlist we need to set up \enitdp@#1

```

216 \cs_set_eq:cc { enitdp@#1 } { g__block_ #1 _depth_int }
217 %
218 }
219 \cs_new:Npn \__block_setup_enum_list_instance:nnn #1#2#3 {
220   \IfInstanceExistsF{list}{#2-#3}
221   { \DeclareInstanceCopy {list} {#2-#3} {#2-\int_eval:n{#3-1}} }
222   \DeclareInstanceCopy {list} {#1-#3} {#2-#3}
223   \str_case:nnF { #2 }
224   {
225
226     { enumerate }
227     {

```

```

228     \EditInstance {list} {#1-#3} {
229         counter:e      =      #1 \int_to_roman:n {#3}
230         ,item-label:c = label #1 \int_to_roman:n {#3}
231     }
232     \newcounter { #1 \int_to_roman:n {#3} }
233     \tl_new:c { label #1 \int_to_roman:n {#3} }
234     \tl_set:cn { label #1 \int_to_roman:n {#3} } { \arabic* }
235 }
236 { itemize }
237 {
238     \EditInstance {list} {#1-#3} {
239         item-label:c = label #1 \int_to_roman:n {#3}
240     }
241     \tl_new:c { label #1 \int_to_roman:n {#3} }
242     \tl_set:cn { label #1 \int_to_roman:n {#3} } { -- }
243 }
244 }
245 { \ERROR-#2-unknown }
246 }

```

(End of definition for `\newlist`. This function is documented on page ??.)

7.3.9 `\renewlist`

7.3.10 `\setlist`

For emulating `\setlist` I lifted most of the code directly from `enumitem` for now and made only minimal adjustments. Size-dependent settings via `<...>` are not supported so far.

TODO: rewrite in `expl3`. TODO: decide if we want to emulate size-dependent settings

```

247 \ExplSyntaxOff
248 \makeatletter

249 \newcommand\setlist{%
250     \@ifstar{\enit@setlist\@ne}{\enit@setlist\z@}}

251 \def\enit@setlist#1{%
252     \@ifnextchar<%
253         {\enit@setlist@q#1}%
254         {\let\enit@forsize\@empty\enit@setlist@n#1}}

```

Size-dependent settings are not supported at all.

```

255 \def\enit@setlist@q#1<#2>{%
256     \enit@error
257     {Size feature not implemented}%
258     {Size dependent setting is not available in the emulation}%
259     % {Activate this feature with options 'sizes'}%
260     % {Size dependent setting with \string<\string> must be\MessageBreak
261     %     explicitly activated with the package option 'sizes'}}
262 }

263 \def\enit@setlist@n#1{%
264     \@ifnextchar[{\enit@setlist@x#1}{\enit@setlist@i#1\@empty}}

```

```

265 % Let's accept \setlist[]*{}, too, because an error in <=3.5.1
266
267 \def\enit@setlist@x#1[#2]{%
268   \ifstar{\enit@setlist@i\@ne{#2}}{\enit@setlist@i#1{#2}}
269
270 \def\enit@setlist@i#1#2#3{%
271   \let\enit@eltnames\relax
272   \let\enit@b\@empty
273   \let\enit@eltlevels\relax
274   \let\enit@c\@empty
275   \protected@edef\enit@a{#2}%
276   \@for\enit@a:=\enit@a\do{% the 2nd enit@a is first expanded
277     \enit@ifunset{\enitdp{\enit@meaning\enit@a}%
278       {\edef\enit@c{\enit@c\enit@eltlevels{\enit@a}}}%
279       {\edef\enit@b{\enit@b\enit@eltnames{\enit@a}}}%
280     }
281   \ifx\enit@b\@empty
282     \def\enit@b{\enit@eltnames{list}}%
283   \fi
284   \ifx\enit@c\@empty
285     \def\enit@c{\enit@eltlevels{0}}%
286   \fi
287   \def\enit@eltnames##1{%
288     \def\enit@a{##1}%
289     \enit@c}%
290   \def\enit@eltlevels##1{%
291     \enit@saveset\enit@a{##1}#1{#3}}%
292   \enit@b}%
293
294 \let\c@enit@cnt\@tempcnta
295
296 \def\enit@meaning{\expandafter\strip@prefix\meaning}
297
298 \long\def\enit@afterelse#1\else#2\fi{\fi#1}
299 \long\def\enit@afterfi#1\fi{\fi#1}
300
301 \def\enit@error{\PackageError{enumitem}}
302
303 \def\enit@ifunset#1{%
304   \ifcsname#1\endcsname
305     \expandafter\ifx\csname#1\endcsname\relax
306       \enit@afterelse\expandafter\@firstoftwo
307     \else
308       \enit@afterfi\expandafter\@secondoftwo
309     \fi
310   \else
311     \expandafter\@firstoftwo
312   \fi
313 }
314
315 \def\enit@saveset#1#2#3#4{%
316   \setcounter{enit@cnt}{#2}%
317   \ifx\enit@forsize\@empty
318     \ifcase#3%
319       \expandafter
320       \def\csname enit__block#1\romannumeral\c@enit@cnt\endcsname{#4}%

```

```

313 \or
314 \expandafter\let\expandafter\enit@b
315 \csname enit__block#1\romannumeral\c@enit@cnt\endcsname
316 \ifx\enit@b\relax
317 \let\enit@b\@empty
318 \fi
319 \expandafter\def
320 \csname enit__block#1\romannumeral\c@enit@cnt\expandafter\endcsname
321 \expandafter{\enit@b,#4}%
322 \fi
323 \else
324 \ifcase#3%
325 \enit@ifunset{enit__block#1\romannumeral\c@enit@cnt}%
326 {\expandafter\let
327 \csname enit__block#1\romannumeral\c@enit@cnt\endcsname\@empty}%
328 {}%
329 \expandafter\let\expandafter\enit@b
330 \csname enit__block#1\romannumeral\c@enit@cnt __blocksizes\endcsname
331 \ifx\enit@b\relax
332 \let\enit@b\@empty
333 \fi
334 \toks@\expandafter{\enit@b}%
335 \edef\enit@b{\the\toks@\enit@forsize\enit@keys@sizes}%
336 \expandafter\def
337 \csname enit__block#1\romannumeral\c@enit@cnt __blocksizes\expandafter\endcsname
338 \expandafter{\enit@b{#4}}%
339 \else
340 \enit@error{* and \string<\string> are not compatible}%
341 {Use either * or angles, but not both.}%
342 \fi
343 \fi}

```

The enumitem code uses `enitdp@⟨name⟩` to refer to the list level counter of the `⟨name⟩` list. So we do need that around for each standard list and provide one when making a new list with `\newlist`.

TODO: check how to do this when making new lists directly.

```

344 \let\enitdp@enumerate\@enumdepth
345 \let\enitdp@itemize\@itemdepth
346 \let\enitdp@description\@descriptiondepth

```

To put keys set with `\setlist` into the key processing of `blockenv` instances we sneak them in as if they are given in the optional argument. For this we evaluate `\enit__blocklist` (produced by `\setlist{...}`), `\enit__blocklist⟨roman num⟩` (produced by `\setlist[num]{...}`), `\enit__block⟨name⟩` (produced by `\setlist[name]{...}`), and `\enit__block⟨name⟩⟨roman num⟩` (produced by `\setlist[name,num]{...}`) in that order and collect all key settings stored in them and stored them in `\UnusedTemplateKeys` which is then used at the start of the `blockenv` template.

```

347 \AddToHookWithArguments{blockenv}{%

```

If `\enitdp@#1` is undefined the current `blockenv` is not a list, so we bail out.

```

348 \enit@ifunset{enitdp@#1}{}%
349 {%

```

Otherwise put the current list level in \@tempcnta and then evaluate the four storage places for keys.

```

350   \@tempcnta\csname enitdp@#1\endcsname
351   \advance\@tempcnta\@ne
352   \enit@setkeys{list}%
353   \enit@setkeys{list\romannumeral\@tempcnta}%
354   \enit@setkeys{#1}%
355   \enit@setkeys{#1\romannumeral\@tempcnta}%
356   }%
357 }

358 \ExplSyntaxOn

```

Evaluation means checking if \enit__block#1 exists and if so add its content to \UnusedTemplateKeys followed by a comma. Thus, if there aren't any keys then \UnusedTemplateKeys remains empty. Otherwise it ends in a comma so that the blockenv template can safely append any keys from the document to make the full list of keys to be evaluated by the template instance.

```

359 \cs_new_protected:Npn \enit@setkeys #1 {
360   \enit@ifunset {enit__block#1} {}
361   {
362     \tl_set:Nx \UnusedTemplateKeys
363       {
364         \exp_not:o \UnusedTemplateKeys
365         \exp_not:v {enit__block#1}
366       },
367   }
368 }
369 }
370 \makeatother

```

7.3.11 \setlistdepth

7.3.12 \AddEnumerateCounter

7.3.13 \SetEnumitemSize

7.3.14 \restartlist

```

371 \</package>

```