

The latex-lab-enumitem package

Emulating enumitem

L^AT_EX Project*

v0.80i 2026-10-24

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>		yes		see key description
<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!		yes		
labelindent*				behaves like !
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		yes		
labelsep*				behaves like !
labelwidth	label-width			see key description
left				
widest				
widest*				
start	start	yes	?	
resume	resume	yes	?	see key description
resume*				
series				
beginpenalty	begin-penalty	yes	?	
midpenalty	item-penalty	yes	?	
endpenalty	end-penalty	yes	?	
before				
before*				
after				
after*				
first				
first*				
style				
noitemsep	meta-key	yes		
nosep	meta-key	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

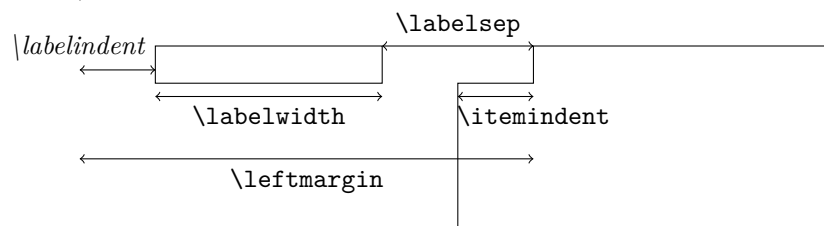
4 Commands

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:

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

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/or declare that another one is calculated by using `!` and `*` as value.

Calculation should obviously happen after all keys are set, but in the template code the values live in different templates: `\leftmargin` is set (and used!) in the block template while the other values are set in the list template.

`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}{}
35     }
36   }
37 }

```

7.1.2 label*

TODO:

7.1.3 ref

This is a rather crude implementation TODO: do more tests.

```

38 \keys_define:nn{template/list/std}
39 {
40   ref .code:n =
41   {
42     \tl_if_empty:NF\l__block_counter_tl
43     {
44       \tl_set:cn{the\l__block_counter_tl}{#1}
45       \tl_set:cn{p@\l__block_counter_tl}{}
46     }
47   }
48 }

```

7.1.4 font, format

```
49 \keys_define:nn{template/item/std}  
50 {  
51   font .meta:n = {label-format={#1{##1}}},  
52   format .meta:n = {label-format={#1{##1}}}  
53 }
```

7.1.5 item/std equivalents

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

```
54 \DeclareTemplateInterfaceEquivalents {item} {std}  
55 { align : alias = label-align }
```

7.1.6 list/std equivalents

midpenalty

```
56 \DeclareTemplateInterfaceEquivalents {list} {std}  
57 {  
58   midpenalty : alias = item-penalty  
59 }
```

7.1.7 block/std equivalents

```
60 \DeclareTemplateInterfaceEquivalents{block}{std}  
61 {
```

topsep

```
62   ,topsep : meta = { begin-vspace={#1}, end-vspace={#1} }
```

partopsep

```
63   , partopsep : meta = { begin-extra-vspace={#1}, end-extra-vspace={#1} }
```

parsep

```
64   , parsep : alias = para-vspace
```

itemsep

```
65   , itemsep : alias = item-vspace
```

rightmargin

```
66   , rightmargin : alias = right-margin
```

listparindent

```
67   , listparindent : alias = para-indent
```

beginpenalty, endpenalty

```
68 , beginpenalty : alias = begin-penalty,
69 , endpenalty : alias = end-penalty,
```

end of block/std equivalents

```
70 }
```

7.1.8 Calculating horizontal values

The following five keys are related to the calculation mentioned in 5

$\backslash\text{labelindent} + \backslash\text{labelwidth} + \backslash\text{labelsep} = \backslash\text{leftmargin} + \backslash\text{itemindent}$ We define a command which will store the value which is to be calculated: We follow `enumitem` and store 0=`labelindent`, 1=`labelwidth`, 2=`labelsep`, 3=`leftmargin`, 4=`itemindent`. We also store if the calculation should take the width of the labels into account. `leftmargin` is a block template key, if we want to set it through calculation we have to process all keys already in the block template key. Note this then affect also nested list, but this is the case with `enumitem` too.

```
71 \tl_new:N \l__block_enumitem_calc_tl
72 \tl_set:Nn \l__block_enumitem_calc_tl{0}
73 \bool_new:N
74 \l__block_enumitem_calc_widest_bool
```

A message for * values, which are not properly supported yet:

```
75 \msg_new:nnn {block}{enumitem-star}{The-key-value~`*~is-not~supported.\\ Value~`!~instead~v
```

leftmargin TODO: * works like ! currently.

```
76 \cs_new:Npn \__block_warning_star_leftmargin:
77 {
78   \msg_warning:nnn {block}{enumitem-star}{leftmargin}
79   \cs_gset_eq:NN \__block_warning_star_leftmargin:\prg_do_nothing:
80 }
81 \keys_define:nn{template/block/std}
82 {
83   leftmargin .code:n =
84   {
85     \str_case:nnF {#1}
86     {
87       {}
88       {
89         \tl_set:Nn \l__block_enumitem_calc_tl{3}
90         \bool_set_false:N \l__block_enumitem_calc_widest_bool
91       }
92       {*}
93       {
94         \tl_set:Nn \l__block_enumitem_calc_tl{3}
95         \bool_set_true:N \l__block_enumitem_calc_widest_bool
96         \__block_warning_star_leftmargin:
97       }
98     }
99     {\setlength{\leftmargin}{#1}}
100   }
101 }
```

itemindent TODO: * works like ! currently.

```
102 \cs_new:Npn\__block_warning_star_itemindent:
103 {
104   \msg_warning:nnn {block}{enumitem-star}{itemindent}
105   \cs_gset_eq:NN \__block_warning_star_itemindent:\prg_do_nothing:
106 }
107 \keys_define:nn{template/block/std}
108 {
109   itemindent .code:n =
110     \str_case:nnF {#1}
111     {
112       {!}
113       {
114         \tl_set:Nn\l__block_enumitem_calc_tl{4}
115         \bool_set_false:N \l__block_enumitem_calc_widest_bool
116       }
117       {*}
118       {
119         \tl_set:Nn\l__block_enumitem_calc_tl{4}
120         \bool_set_true:N \l__block_enumitem_calc_widest_bool
121         \__block_warning_star_itemindent:
122       }
123     }
124     {\setlength\itemindent{#1}}
125 }
```

labelsep, labelsep* TODO: * works like ! currently. Add a warning?

```
126 \cs_new:Npn\__block_warning_star_labelsep:
127 {
128   \msg_warning:nnn {block}{enumitem-star}{leftmargin}
129   \cs_gset_eq:NN \__block_warning_star_labelsep:\prg_do_nothing:
130 }
131 \keys_define:nn{template/block/std}
132 {
133   labelsep .code:n =
134     \str_case:nnF {#1}
135     {
136       {!}
137       {
138         \tl_set:Nn\l__block_enumitem_calc_tl{2}
139         \bool_set_false:N \l__block_enumitem_calc_widest_bool
140       }
141       {*}
142       {
143         \tl_set:Nn\l__block_enumitem_calc_tl{2}
144         \bool_set_true:N \l__block_enumitem_calc_widest_bool
145         \__block_warning_star_labelsep:
146       }
147     }
148     {\setlength\labelsep{#1}}
149 }
```

labelwidth TODO: * is the same as !.

```

150 \cs_new:Npn\__block_warning_star_labelwidth:
151 {
152   \msg_warning:nnn {block}{enumitem-star}{leftmargin}
153   \cs_gset_eq:NN \__block_warning_star_labelwidth:\prg_do_nothing:
154 }
155 \keys_define:nn{template/block/std}
156 {
157   labelwidth .code:n =
158   \str_case:nnF {#1}
159   {
160     {!}
161     {
162       \tl_set:Nn\l__block_enumitem_calc_tl{1}
163       \bool_set_false:N \l__block_enumitem_calc_widest_bool
164     }
165     {*}
166     {
167       \tl_set:Nn\l__block_enumitem_calc_tl{1}
168       \bool_set_true:N \l__block_enumitem_calc_widest_bool
169       \__block_warning_star_labelwidth:
170     }
171   }
172   {\setlength\labelwidth{#1}}
173 }

```

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

```

174 \cs_new:Npn\__block_warning_star_labelindent:
175 {
176   \msg_warning:nnn {block}{enumitem-star}{labelindent}
177   \cs_gset_eq:NN \__block_warning_star_labelindent:\prg_do_nothing:
178 }
179 \newlength\labelindent
180 \keys_define:nn{template/block/std}
181 {
182   labelindent .code:n =
183   \str_case:nnF {#1}
184   {
185     {!}
186     {
187       \tl_set:Nn\l__block_enumitem_calc_tl{0}
188       \bool_set_false:N \l__block_enumitem_calc_widest_bool
189     }
190     {*}
191     {
192       \tl_set:Nn\l__block_enumitem_calc_tl{0}
193       \bool_set_true:N \l__block_enumitem_calc_widest_bool
194       \__block_warning_star_labelindent:
195     }
196   }
197   {\setlength\labelindent{#1}}
198 }

```

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

```

199 \cs_set:Npn \__block_recalculate_list_values:
200 {
201   \int_case:nn { \l__block_enumitem_calc_tl }
202   {
203     {0} % 0=labelindent
204     {\setlength\labelindent{\dim_eval:n{\leftmargin+\itemindent-\labelsep-\labelwidth}}}
205     {1} % 1=labelwidth
206     {\setlength\labelwidth{\dim_eval:n{\leftmargin+\itemindent-\labelindent-
\labelsep}}}
207     {2} % 2=labelsep
208     {\setlength\labelsep{\dim_eval:n{\leftmargin+\itemindent-\labelindent-\labelwidth}}}
209     {3} % 3=leftmargin
210     {\setlength\leftmargin{\dim_eval:n{\labelindent+\labelwidth+\labelsep-\itemindent}}}
211     {4} % 4=itemindent
212     {\setlength\itemindent{\dim_eval:n{\labelindent+\labelwidth+\labelsep-\leftmargin}}}
213   }
214 }

```

7.1.9 left

TODO:

7.1.10 widest, widest*

TODO:

7.1.11 start

TODO: Test with setlist

7.1.12 resume

TODO: Test with setlist.

The behavior of the key is different to enumitem, where grouping of the environments matters: in enumitem a enumerate that is e.g. in quote environment can not be resumed outside of the quote. With the L^AT_EX code grouping does not matter.

7.1.13 resume*

TODO: decide if it should be implemented

7.1.14 series

TODO: decide if it should be implemented

7.1.15 before, before*

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

7.1.16 after, after*

TODO: implement with hooks?

7.1.17 first, first*

TODO: implement with hooks?

7.1.18 style

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

7.1.19 noitemsep, nosep

```
215 \keys_define:nn{template/blockenv/std}  
216 {  
217   nosep .meta:n =  
218   {  
219     begin-vspace=0pt,  
220     begin-extra-vspace=0pt,  
221     end-vspace=0pt,  
222     end-extra-vspace=0pt,  
223     item-vspace=0pt,  
224     para-vspace=0pt  
225   }  
226   ,noitemsep .meta:n =  
227   {  
228     item-vspace=0pt,  
229     para-vspace=0pt  
230   }  
231 }  
232
```

7.1.20 wide

TODO: calculated value should be delayed ...

```
233 \keys_define:nn{template/blockenv/std}  
234 {  
235   wide .meta:n =  
236   {  
237     label-align=left,  
238     para-indent=#1,  
239     left-margin=0pt,  
240     label-width=0pt,  
241     item-indent=\dimeval{#1+\labelsep} %should be delayed ....  
242   },  
243   wide .default:n = \parindent  
244 }
```

7.1.21 itemjoin, itemjoin*, afterlabel

TODO

7.1.22 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 key `labelindent`

7.3.4 \EnumitemId

7.3.5 \SetEnumitemKey

The `\SetEnumitemKey` defines shortcuts. We can define 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).

```
245 \NewDocumentCommand \SetEnumitemKey {mm} {  
246   \keys_define:nn { template/block/std }  
247     { #1 .meta:n = { #2 } }  
248 }
```

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`

```

249 \NewDocumentCommand\newlist{mmm} %name, type, number
250 {
251   \str_case:nnF{#2}
252   {
253     {itemize}    {\__block_setup_itemize:nn{#1}{#3}}
254     {enumerate} {\__block_setup_enumerate:nn{#1}{#3}}
255     {description}{\__block_setup_description:nn{#1}{#3}}
256   }

```

TODO: message

```

257   {\typeout{unknown~list~type~#2}
258 }

```

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

```

259 }
260 %

```

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

`\__block_setup_itemize:nn`

```

261 \cs_new_protected:Npn \__block_setup_itemize:nn #1#2 %#1 name of new list, #2 max levels
262 {
263   \DeclareInstanceCopy{blockenv}{#1}{itemize}
264   \EditInstance{blockenv}{#1}{inner-instance=#1}
265   \NewDocumentEnvironment{#1}{0{}}
266   { \SimpleBlockEnv {#1} {max-inner-levels= \int_min:nn{\c@maxblocklevels}{#2},##1} }
267   { \BlockEnvEnd }
268   \int_step_inline:nnn{1}{\int_min:nn{\c@maxblocklevels}{#2}}
269   {
270     \IfInstanceExistsTF{list}{itemize-##1}
271     {
272       \DeclareInstanceCopy{list}{#1-##1}{itemize-##1}
273     }

```

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

```

274   {
275     \ExpandArgs{c}
276     \providecommand{labelitem\int_to_roman:n{##1}}{\labelitemi}
277     \DeclareInstance{list}{#1-##1}{std}
278     { item-label = \use:c{labelitem\int_to_roman:n{##1}}}
279   }
280 }
281 }

```

(End of definition for __block_setup_itemize:nn.)

`\__block_setup_description:nn`

```

282 \cs_new_protected:Npn \__block_setup_description:nn #1#2
283 {
284   \DeclareInstanceCopy{blockenv}{#1}{description}
285   \EditInstance{blockenv}{#1}{inner-instance=#1}
286   \DeclareInstanceCopy{list}{#1}{description}
287   \NewDocumentEnvironment{#1}{0{}}
288   { \SimpleBlockEnv{#1} {max-inner-levels= \int_min:nn{\c@maxblocklevels}{#2},##1} }
289   { \BlockEnvEnd }
290 }

```

(End of definition for __block_setup_description:nn.)

`\__block_setup_enumerate:nn`

```

291 \cs_new_protected:Npn \__block_setup_enumerate:nn #1#2
292 {
293   \DeclareInstanceCopy{blockenv}{#1}{enumerate}
294   \EditInstance{blockenv}{#1}{inner-instance=#1}
295   \NewDocumentEnvironment{#1}{0{}}
296   { \SimpleBlockEnv {#1} {max-inner-levels= #2,##1} }
297   { \BlockEnvEnd }
298   \int_step_inline:nnn{1}{#2}
299   {
300     \newcounter{#1\int_to_roman:n{##1}}
301     \ExpandArgs{c}
302     \newcommand{label#1\int_to_roman:n{##1}}
303       {\arabic{#1\int_to_roman:n{##1}}.}
304     \DeclareInstance{list}{#1-##1}{std}
305     {
306       item-label = \use:c{label#1\int_to_roman:n{##1}} ,
307       counter    = {#1\int_to_roman:n{##1}}
308     }
309   }
310 }

```

(End of definition for __block_setup_enumerate:nn.)

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

```

311 \cs_set_protected:Npn \newlist #1#2#3 {
312   \DeclareDocumentEnvironment {#1} { !0{ } }
313   { \SimpleBlockEnv {#1}{##1} } { \BlockEnvEnd }
314   %
315   \DeclareInstanceCopy {blockenv} {#1} {#2}
316   %
317   % If #3 > maxblocklevels add additional std-list instances as necessary
318   %
319   \int_compare:nNnT { \value{maxblocklevels} } < {#3}
320   {
321     \int_step_inline:nnn { \value{maxblocklevels} + 1 } {#3}
322     {
323       \DeclareInstanceCopy {block} {std-list-##1} {std-list-\int_eval:n{ ##1 - 1 }}
324       % not sure if we should set up those as well ...
325       \DeclareInstanceCopy {block} {std-display-##1} {std-display-1}
326       \DeclareInstanceCopy {block} {quote-##1} {quote-1}
327       \DeclareInstanceCopy {block} {quotation-##1} {quotation-1}

```

```

328 \DeclareInstanceCopy {block} {verbatim-##1} {verbatim-1}
329 }
330 \setcounter{maxblocklevels}{#3}
331 }
332 %
333 \IfInstanceExistsTF{list}{#2} % no levels (as with description)
334 {
335 \EditInstance {blockenv} {#1} {
336 name = #1
337 ,inner-instance = #1
338 }
339 %
340 \DeclareInstanceCopy {list} {#1} {#2}
341 }
342 {
343 \int_new:c { g__block_ #1 _depth_int }
344 %
345 \EditInstance {blockenv} {#1} {
346 name = #1
347 ,inner-instance = #1
348 ,max-inner-levels = #3
349 ,inner-level-counter:c = g__block_ #1 _depth_int
350 }
351 %
352 \int_step_inline:nn {#3}
353 { \__block_setup_enum_list_instance:nnn {#1}{#2}{##1} }
354 }
355 %

```

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

```

356 \cs_set_eq:cc { enitdp@#1 } { g__block_ #1 _depth_int }
357 %
358 }
359 \cs_new:Npn \__block_setup_enum_list_instance:nnn #1#2#3 {
360 \IfInstanceExistsF{list}{#2-#3}
361 { \DeclareInstanceCopy {list} {#2-#3} {#2-\int_eval:n{#3-1}} }
362 \DeclareInstanceCopy {list} {#1-#3} {#2-#3}
363 \str_case:nnF { #2 }
364 {
365
366 { enumerate }
367 {
368 \EditInstance {list} {#1-#3} {
369 counter:e = #1 \int_to_roman:n {#3}
370 ,item-label:c = label #1 \int_to_roman:n {#3}
371 }
372 \newcounter { #1 \int_to_roman:n {#3} }
373 \tl_new:c { label #1 \int_to_roman:n {#3} }
374 \tl_set:cn { label #1 \int_to_roman:n {#3} } { \arabic* }
375 }
376 { itemize }
377 {
378 \EditInstance {list} {#1-#3} {
379 item-label:c = label #1 \int_to_roman:n {#3}

```

```

380     }
381     \tl_new:c { label #1 \int_to_roman:n {#3} }
382     \tl_set:cn { label #1 \int_to_roman:n {#3} } { -- }
383 }

```

Nothing special to do for `description` but we need a case for it nonetheless to avoid generating an error.

```

384     { description } { }
385 }
386 { \ERROR-#2-unknown }
387 }

```

(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

```

388 \ExplSyntaxOff
389 \makeatletter

390 \newcommand\setlist{%
391   \@ifstar{\enit@setlist\@ne}{\enit@setlist\z@}}

392 \def\enit@setlist#1{%
393   \@ifnextchar<%
394     {\enit@setlist@q#1}%
395     {\let\enit@forsize\@empty\enit@setlist@n#1}}

```

Size-dependent settings are not supported at all.

```

396 \def\enit@setlist@q#1<#2>{%
397   \enit@error
398     {Size feature not implemented}%
399     {Size dependent setting is not available in the emulation}%
400   %   {Activate this feature with options 'sizes'}%
401   %   {Size dependent setting with \string<\string> must be\MessageBreak
402   %     explicitly activated with the package option 'sizes'}}
403 }

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

406 % Let's accept \setlist[]*{}, too, because an error in <=3.5.1
407
408 \def\enit@setlist@x#1[#2]{%
409   \@ifstar{\enit@setlist@i\@ne{#2}}{\enit@setlist@i#1{#2}}}

410 \def\enit@setlist@i#1#2#3{%
411   \let\enit@eltnames\relax
412   \let\enit@b\@empty
413   \let\enit@eltlevels\relax

```

```

414 \let\enit@c\@empty
415 \protected@edef\enit@a{#2}%
416 \@for\enit@a:=\enit@a\do{% the 2nd enit@a is first expanded
417   \enit@ifunset{enitdp@\enit@meaning\enit@a}%
418   {\edef\enit@c{\enit@c\enit@eltlevels{\enit@a}}}%
419   {\edef\enit@b{\enit@b\enit@eltnames{\enit@a}}}%
420 \ifx\enit@b\@empty
421   \def\enit@b{\enit@eltnames{list}}%
422 \fi
423 \ifx\enit@c\@empty
424   \def\enit@c{\enit@eltlevels{0}}%
425 \fi
426 \def\enit@eltnames##1{%
427   \def\enit@a{##1}%
428   \enit@c}%
429 \def\enit@eltlevels##1{%
430   \enit@saveset\enit@a{##1}#1{#3}}%
431 \enit@b}%

432 \let\c@enit@cnt\@tempcnta

433 \def\enit@meaning{\expandafter\strip@prefix\meaning}

434 \long\def\enit@afterelse#1\else#2\fi{\fi#1}
435 \long\def\enit@afterfi#1\fi{\fi#1}

436 \def\enit@error{\PackageError{enumitem}}

437 \def\enit@ifunset#1{%
438   \ifcsname#1\endcsname
439   \expandafter\ifx\csname#1\endcsname\relax
440     \enit@afterelse\expandafter\@firstoftwo
441   \else
442     \enit@afterfi\expandafter\@secondoftwo
443   \fi
444 \else
445   \expandafter\@firstoftwo
446 \fi
447 }

448 \def\enit@saveset#1#2#3#4{%
449   \setcounter{enit@cnt}{#2}%
450   \ifx\enit@forsize\@empty
451     \ifcase#3%
452       \expandafter
453       \def\csname enit__block#1\romannumeral\c@enit@cnt\endcsname{#4}%
454     \or
455       \expandafter\let\expandafter\enit@b
456       \csname enit__block#1\romannumeral\c@enit@cnt\endcsname
457       \ifx\enit@b\relax
458         \let\enit@b\@empty
459       \fi
460     \expandafter\def
461     \csname enit__block#1\romannumeral\c@enit@cnt\expandafter\endcsname
462     \expandafter{\enit@b,#4}%

```

```

463 \fi
464 \else
465 \ifcase#3%
466 \enit@ifunset{enit__block#1\romannumeral\c@enit@cnt}%
467 {\expandafter\let
468 \csname enit__block#1\romannumeral\c@enit@cnt\endcsname\@empty}%
469 {}%
470 \expandafter\let\expandafter\enit@b
471 \csname enit__block#1\romannumeral\c@enit@cnt __blocksizes\endcsname
472 \ifx\enit@b\relax
473 \let\enit@b\@empty
474 \fi
475 \toks@\expandafter{\enit@b}%
476 \edef\enit@b{\the\toks@\enit@forsize\enit@keys@sizes}%
477 \expandafter\def
478 \csname enit__block#1\romannumeral\c@enit@cnt __blocksizes\expandafter\endcsname
479 \expandafter{\enit@b{#4}}%
480 \else
481 \enit@error{* and \string<\string> are not compatible}%
482 {Use either * or angles, but not both.}%
483 \fi
484 \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.

```

485 \let\enitdp@enumerate\@enumdepth
486 \let\enitdp@itemize\@itemdepth
487 \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.

```

488 \AddToHookWithArguments{blockenv}{%

```

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

```

489 \enit@ifunset{enitdp@#1}{}%
490 {%

```

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

```

491 \@tempcnta\csname enitdp@#1\endcsname
492 \advance\@tempcnta\@ne
493 \enit@setkeys{list}%
494 \enit@setkeys{list\romannumeral\@tempcnta}%
495 \enit@setkeys{#1}%
496 \enit@setkeys{#1\romannumeral\@tempcnta}%
497 }%
498 }

```

499 \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.

```
500 \cs_new_protected:Npn \enit@setkeys #1 {  
501   \enit@ifunset {enit__block#1} {}  
502   {  
503     \clist_put_right:Nv \UnusedTemplateKeys { enit__block #1 }  
504   }  
505 }  
506 \makeatother
```

7.3.11 `\setlistdepth`

7.3.12 `\AddEnumerateCounter`

7.3.13 `\SetEnumitemSize`

7.3.14 `\restartlist`

507 `\</package>`