The latex-lab-enumitem package
Emulating enumitem

LaTeX Project*

v0.80b 2026-01-16

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 dependant parameters.

With the new LaTeXimplementation 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 LaTeX 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	LaTeX key	opt. arg.	`\setlist`	comment
`label`	(related: `item-label`)			see key description
`label*`				
`ref`				
`font,format`	`label-format`	yes		see key description!!
`format`				

*Initial implementation done by Ulrike Fischer

Table 1: List of enumitem keys

enumitem key	LaTeX key	opt. arg.	\setlist	comment
align	label-align	yes		see key description
topsep	begin-vspace	yes		
partopsep	begin-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	?	
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 planed; 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

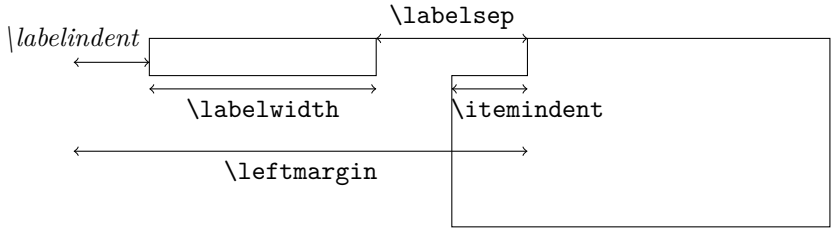
4 Commands

Table 3: List of enumitem user commands

name	emulated	comment
`\SetLabelAlign`		
`\DrawEnumitemLabel`		
`\labelindent`		
`\EnumitemId`		
`\SetEnumitemKey`		
`\SetEnumitemValue`		
`\SetEnumerateShortLabel`		
`\newlist`	yes	
`\renewlist`		
`\setlist`		
`\setlistdepth`		
`\AddEnumerateCounter`		
`\restartlist`		
`\SetEnumitemSize`		

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:
`\labelindent + \labelwidth + \labelsep = \leftmargin + \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 dependant 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

⟨*package⟩

7 Implementation

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

7.1 Key emulation

7.1.1 `label`

The key `item-label` from the new LaTeX 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.

```
⟨@@=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.

```
\clist_new:N  \l__block_normalize_cnt_clist
\clist_set:Nn \l__block_normalize_cnt_clist
              {\alph,\Alph,\roman,\Roman,\arabic,\fnsymbol}
```

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

```
\cs_new:Npn\__block_normalize_label:N #1
 {
    \clist_map_inline:Nn\l__block_normalize_cnt_clist
     {
```

```
      \tl_replace_all:Nne#1 {##1*}{\exp_not:N##1{\l__block_counter_tl}}
    }
 }
\cs_generate_variant:Nn\__block_normalize_label:N{c}
\keys_define:nn{template/list/std}
 {
   label .code:n =
    {
     \tl_if_empty:NTF\l__block_counter_tl
      {
        \tl_if_eq:NnTF\l__block_inner_instance_tl{itemize}
         {
          \tl_set:cn{labelitem\int_to_roman:n{\l__block_inner_level_counter_tl}}{#1}
         }
         {
          \tl_set:cn{label\l__block_inner_instance_tl\int_to_roman:n{\l__block_inner_level_co
         }
      }
      {
       \tl_set:cn{label\l__block_counter_tl}{#1}
        \__block_normalize_label:c{label\l__block_counter_tl}
       \tl_set_eq:cc{the\l__block_counter_tl}{label\l__block_counter_tl}
       \tl_set:cn{p@\l__block_counter_tl}{}
      }
    }
 }
```

7.1.2 label*

TODO:

7.1.3 ref

TODO:

7.1.4 font, format

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

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

7.1.6 topsep

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

7.1.7 partopsep

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

7.1.8 parsep

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

7.1.9 itemsep

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

7.1.10 leftmargin

TODO: handle special values ! and *.
Special values do not work with LaTeX key at the moment as it is a register!

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

7.1.11 rightmargin

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

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

7.1.12 listparindent

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

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

7.1.13 itemindent

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

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

7.1.14 labelsep, labelsep*

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

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

7.1.15 labelwidth

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

```
\keys_define:nn{template/list/std}
  { 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 a 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

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

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

```
\keys_define:nn{template/blockenv/std}
 {
   nosep .meta:n =
    {
      begin-extra-vspace=0pt,
      begin-vspace=0pt,
      item-vspace=0pt,
      para-vspace=0pt
    }
 }
\keys_define:nn{template/blockenv/std}
 {
   noitemsep .meta:n =
    {
      item-vspace=0pt,
      para-vspace=0pt
    }
 }

```

7.1.29 wide

TODO: calculated value should be delayed ...

```
\keys_define:nn{template/blockenv/std}
 {
  wide .meta:n =
   {
    label-align=left,
    para-indent=#1,
    left-margin=0pt,
    label-width=0pt,
    item-indent=\dimeval{#1+\labelsep} %should be delayed ....
   },
  wide .default:n = \parindent
 }
```

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).

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

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.

\newlist

```
\NewDocumentCommand\newlist{mmm} %name, type, number
 {
   \str_case:nnF{#2}
    {
     {itemize}    {\__block_setup_itemize:nn{#1}{#3}}
     {enumerate}  {\__block_setup_enumerate:nn{#1}{#3}}
     {description}{\__block_setup_description:nn{#1}{#3}}
    }
```

TODO: message

```
    {\typeout{unknown~list~type~#2}}
 }
```

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

__block_setup_itemize:nn

```
\cs_new_protected:Npn \__block_setup_itemize:nn #1#2 %#1 name of new list, #2 max levels
 {
    \DeclareInstanceCopy{blockenv}{#1}{itemize}
    \EditInstance{blockenv}{#1}{inner-instance=#1}
    \NewDocumentEnvironment{#1}{!O{}}
     { \SimpleBlockEnv {#1} {max-inner-levels= \int_min:nn{\c@maxblocklevels}{#2},##1} }
     { \BlockEnvEnd }
    \int_step_inline:nnn{1}{\int_min:nn{\c@maxblocklevels}{#2}}
     {
       \IfInstanceExistsTF{list}{itemize-##1}
        {
          \DeclareInstanceCopy{list}{#1-##1}{itemize-##1}
        }
```

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

```
        {
          \ExpandArgs{c}
           \providecommand{labelitem\int_to_roman:n{##1}}{\labelitemi}
          \DeclareInstance{list}{#1-##1}{std}
           { item-label = \use:c{labelitem\int_to_roman:n{##1}}}
        }
     }
 }
```

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

__block_setup_description:nn

```
\cs_new_protected:Npn \__block_setup_description:nn #1#2
  {
    \DeclareInstanceCopy{blockenv}{#1}{description}
    \EditInstance{blockenv}{#1}{inner-instance=#1}
    \DeclareInstanceCopy{list}{#1}{description}
    \NewDocumentEnvironment{#1}{!O{}}
     { \SimpleBlockEnv{#1} {max-inner-levels= \int_min:nn{\c@maxblocklevels}{#2},##1} }
     { \BlockEnvEnd }
  }
```

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

__block_setup_enumerate:nn

```
\cs_new_protected:Npn \__block_setup_enumerate:nn #1#2
  {
    \DeclareInstanceCopy{blockenv}{#1}{enumerate}
    \EditInstance{blockenv}{#1}{inner-instance=#1}
    \NewDocumentEnvironment{#1}{!O{}}
     { \SimpleBlockEnv {#1} {max-inner-levels= #2,##1} }
     { \BlockEnvEnd }
    \int_step_inline:nnn{1}{#2}
     {
       \newcounter{#1\int_to_roman:n{##1}}
       \ExpandArgs{c}
         \newcommand{label#1\int_to_roman:n{##1}}
                    {\arabic{#1\int_to_roman:n{##1}}.}
       \DeclareInstance{list}{#1-##1}{std}
         {
           item-label = \use:c{label#1\int_to_roman:n{##1}} ,
           counter    = {#1\int_to_roman:n{##1}}
         }
     }
  }
```

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

7.3.9 \renewlist

7.3.10 \setlist

```
\NewDocumentCommand\setlist{O{}m}{} %dummy for now
```

7.3.11 \setlistdepth

7.3.12 \AddEnumerateCounter

7.3.13 \SetEnumitemSize

7.3.14 \restartlist

⟨/package⟩