# The latex-lab-enumitem package
# Emulating enumitem

LaTeX 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 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` | | yes | | see key description |
| `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 + 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 |

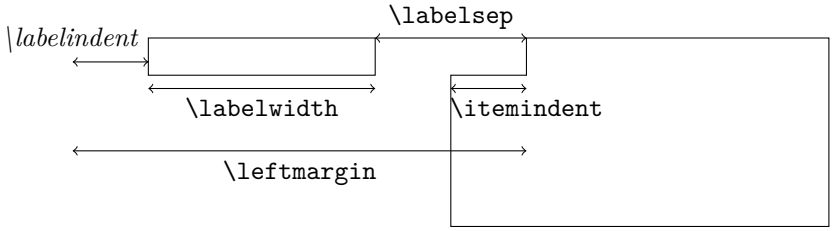
# 4 Commands

Table 3: List of enumitem user commands

| name | emulated | comment |
|---|---|---|
| `\SetLabelAlign` | | |
| `\DrawEnumitemLabel` | | |
| `\labelindent` | | |
| `\EnumitemId` | | |
| `\SetEnumitemKey` | yes | |
| `\SetEnumitemValue` | | |
| `\SetEnumerateShortLabel` | | |
| `\newlist` | yes | |
| `\renewlist` | | |
| `\setlist` | yes | no size-dependent settings |
| `\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/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

```
⟨*package⟩
```

# 7 Implementation

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

## 7.1 Key emulation

### 7.1.1 `label`

check all this code

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`

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

```
\keys_define:nn{template/list/std}
 {
   ref .code:n =
    {
     \tl_if_empty:NF\l__block_counter_tl
      {
       \tl_set:cn{the\l__block_counter_tl}{#1}
       \tl_set:cn{p@\l__block_counter_tl}{}
      }
    }
 }
```

### 7.1.4 font, format

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

### 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.

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

### 7.1.6 list/std equivalents

**midpenalty**

```
\DeclareTemplateInterfaceEquivalents {list} {std}
  {
    midpenalty   : alias   = item-penalty
  }
```

### 7.1.7 block/std equivalents

```
\DeclareTemplateInterfaceEquivalents{block}{std}
 {
```

**topsep**

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

**partopsep**

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

**parsep**

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

**itemsep**

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

**rightmargin**

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

**listparindent**

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

**beginpenalty, endpenalty**

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

end of block/std equivalents

```
}
```

### 7.1.8 Calculating horizontal values

The following five keys are related to the calculation mentioned in 5 `\labelindent + \labelwidth + \labelsep = \leftmargin + \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.

```
\tl_new:N    \l__block_enumitem_calc_tl
\tl_set:Nn   \l__block_enumitem_calc_tl{0}
\bool_new:N
 \l__block_enumitem_calc_widest_bool
```

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

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

**leftmargin** TODO: * works like ! currently.

```
\cs_new:Npn\__block_warning_star_leftmargin:
 {
   \msg_warning:nnn {block}{enumitem-star}{leftmargin}
   \cs_gset_eq:NN \__block_warning_star_leftmargin:\prg_do_nothing:
 }
\keys_define:nn{template/block/std}
   {
     leftmargin .code:n =
     {
       \str_case:nnF {#1}
        {
          {!}
           {
             \tl_set:Nn\l__block_enumitem_calc_tl{3}
             \bool_set_false:N \l__block_enumitem_calc_widest_bool
           }
          {*}
           {
            \tl_set:Nn\l__block_enumitem_calc_tl{3}
            \bool_set_true:N \l__block_enumitem_calc_widest_bool
            \__block_warning_star_leftmargin:
           }
        }
       {\setlength{\leftmargin}{#1}}
     }
   }
```

**itemindent** TODO: * works like ! currently.

```
\cs_new:Npn\__block_warning_star_itemindent:
 {
   \msg_warning:nnn {block}{enumitem-star}{itemindent}
   \cs_gset_eq:NN \__block_warning_star_itemindent:\prg_do_nothing:
 }
\keys_define:nn{template/block/std}
   {
    itemindent .code:n =
     \str_case:nnF {#1}
      {
        {!}
         {
           \tl_set:Nn\l__block_enumitem_calc_tl{4}
           \bool_set_false:N \l__block_enumitem_calc_widest_bool
         }
        {*}
         {
          \tl_set:Nn\l__block_enumitem_calc_tl{4}
          \bool_set_true:N \l__block_enumitem_calc_widest_bool
          \__block_warning_star_itemindent:
         }
      }
      {\setlength\itemindent{#1}}
   }
```

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

```
\cs_new:Npn\__block_warning_star_labelsep:
 {
   \msg_warning:nnn {block}{enumitem-star}{leftmargin}
   \cs_gset_eq:NN \__block_warning_star_labelsep:\prg_do_nothing:
 }
\keys_define:nn{template/block/std}
  {
    labelsep .code:n =
     \str_case:nnF {#1}
      {
        {!}
         {
           \tl_set:Nn\l__block_enumitem_calc_tl{2}
           \bool_set_false:N \l__block_enumitem_calc_widest_bool
         }
        {*}
         {
          \tl_set:Nn\l__block_enumitem_calc_tl{2}
          \bool_set_true:N \l__block_enumitem_calc_widest_bool
          \__block_warning_star_labelsep:
         }
      }
      {\setlength\labelsep{#1}}
  }
```

**labelwidth** TODO: * is the same as !.

```
\cs_new:Npn\__block_warning_star_labelwidth:
 {
   \msg_warning:nnn {block}{enumitem-star}{leftmargin}
   \cs_gset_eq:NN \__block_warning_star_labelwidth:\prg_do_nothing:
 }
\keys_define:nn{template/block/std}
  {
    labelwidth .code:n =
     \str_case:nnF {#1}
      {
        {!}
         {
           \tl_set:Nn\l__block_enumitem_calc_tl{1}
           \bool_set_false:N \l__block_enumitem_calc_widest_bool
         }
        {*}
         {
          \tl_set:Nn\l__block_enumitem_calc_tl{1}
          \bool_set_true:N \l__block_enumitem_calc_widest_bool
          \__block_warning_star_labelwidth:
         }
      }
     {\setlength\labelwidth{#1}}
  }
```

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

```
\cs_new:Npn\__block_warning_star_labelindent:
 {
   \msg_warning:nnn {block}{enumitem-star}{labelindent}
   \cs_gset_eq:NN \__block_warning_star_labelindent:\prg_do_nothing:
 }
\newlength\labelindent
\keys_define:nn{template/block/std}
  {
    labelindent .code:n =
     \str_case:nnF {#1}
      {
        {!}
         {
           \tl_set:Nn\l__block_enumitem_calc_tl{0}
           \bool_set_false:N \l__block_enumitem_calc_widest_bool
         }
        {*}
         {
          \tl_set:Nn\l__block_enumitem_calc_tl{0}
          \bool_set_true:N \l__block_enumitem_calc_widest_bool
          \__block_warning_star_labelindent:
         }
       }
      {\setlength\labelindent{#1}}
  }
```

**The calculation** `\labelindent + \labelwidth + \labelsep = \leftmargin + \itemindent`

```
\cs_set:Npn \__block_recalculate_list_values:
  {
    \int_case:nn { \l__block_enumitem_calc_tl }
    {
      {0} % 0=labelindent
       {\setlength\labelindent{\dim_eval:n{\leftmargin+\itemindent-\labelsep-\labelwidth}}}
      {1} % 1=labelwidth
       {\setlength\labelwidth{\dim_eval:n{\leftmargin+\itemindent-\labelindent-
\labelsep}}}
      {2} % 2=labelsep
       {\setlength\labelsep{\dim_eval:n{\leftmargin+\itemindent-\labelindent-\labelwidth}}}
      {3} % 3=leftmargin
       {\setlength\leftmargin{\dim_eval:n{\labelindent+\labelwidth+\labelsep-\itemindent}}}
      {4} % 4=itemindent
       {\setlength\itemindent{\dim_eval:n{\labelindent+\labelwidth+\labelsep-\leftmargin}}}
    }
  }
```

#### 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 LaTeX 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

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

```

#### 7.1.20 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
 }
```

#### 7.1.21 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 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.

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

```
\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}
 }
```

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

```
 }
%
```

(*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`.)

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

```
\cs_set_protected:Npn \newlist #1#2#3 {
  \DeclareDocumentEnvironment {#1} { !O{} }
    { \SimpleBlockEnv {#1}{##1} } { \BlockEnvEnd }
%
  \DeclareInstanceCopy {blockenv} {#1} {#2}
%
%  If #3 > maxblocklevels add additional std-list instances as necessary
%
  \int_compare:nNnT { \value{maxblocklevels} } < {#3}
    {
      \int_step_inline:nnn { \value{maxblocklevels} + 1 } {#3}
        {
   \DeclareInstanceCopy {block} {std-list-##1} {std-list-\int_eval:n{ ##1 - 1 }}
% not sure if we should set up those as well ...
   \DeclareInstanceCopy {block} {std-display-##1} {std-display-1}
   \DeclareInstanceCopy {block} {quote-##1}       {quote-1}
   \DeclareInstanceCopy {block} {quotation-##1}   {quotation-1}
```

```
   \DeclareInstanceCopy {block} {verbatim-##1}    {verbatim-1}
        }
      \setcounter{maxblocklevels}{#3}
    }
%
  \IfInstanceExistsTF{list}{#2}   % no levels (as with description)
    {
      \EditInstance {blockenv} {#1} {
   name                  = #1
        ,inner-instance        = #1
      }
%
      \DeclareInstanceCopy {list} {#1} {#2}
    }
    {
      \int_new:c { g__block_ #1 _depth_int }
%
      \EditInstance {blockenv} {#1} {
    name                  = #1
        ,inner-instance        = #1
        ,max-inner-levels      = #3
        ,inner-level-counter:c = g__block_ #1 _depth_int
      }
%
      \int_step_inline:nn {#3}
         { \__block_setup_enum_list_instance:nnn {#1}{#2}{##1} }
    }
%
```

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

```
  \cs_set_eq:cc { enitdp@#1 } { g__block_ #1 _depth_int }
%
}

\cs_new:Npn \__block_setup_enum_list_instance:nnn #1#2#3 {
  \IfInstanceExistsF{list}{#2-#3}
    { \DeclareInstanceCopy {list} {#2-#3} {#2-\int_eval:n{#3-1}} }
  \DeclareInstanceCopy {list} {#1-#3} {#2-#3}
  \str_case:nnF { #2 }
    {

      { enumerate }
        {
          \EditInstance {list} {#1-#3} {
            counter:e    =       #1 \int_to_roman:n {#3}
           ,item-label:c = label #1 \int_to_roman:n {#3}
          }
          \newcounter { #1 \int_to_roman:n {#3} }
   \tl_new:c  { label #1 \int_to_roman:n {#3} }
          \tl_set:cn { label #1 \int_to_roman:n {#3} } { \arabic* }
        }
      { itemize }
        {
          \EditInstance {list} {#1-#3} {
             item-label:c = label #1 \int_to_roman:n {#3}
```

```
            }
        \tl_new:c  { label #1 \int_to_roman:n {#3} }
            \tl_set:cn { label #1 \int_to_roman:n {#3} } { -- }
 }
```

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

```
       { description } { }
     }
     { \ERROR-#2-unknown }
}
```

(*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

```
\ExplSyntaxOff
\makeatletter

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

\def\enit@setlist#1{%
  \@ifnextchar<%
    {\enit@setlist@q#1}%
    {\let\enit@forsize\@empty\enit@setlist@n#1}}
```

Size-dependent settings are not supported at all.

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

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

% Let's accept \setlist[]*{}, too, because an error in <=3.5.1

\def\enit@setlist@x#1[#2]{%
  \@ifstar{\enit@setlist@i\@ne{#2}}{\enit@setlist@i#1{#2}}}

\def\enit@setlist@i#1#2#3{%
  \let\enit@eltnames\relax
  \let\enit@b\@empty
  \let\enit@eltlevels\relax
```

```
  \let\enit@c\@empty
  \protected@edef\enit@a{#2}%
  \@for\enit@a:=\enit@a\do{% the 2nd enit@a is first expanded
    \enit@ifunset{enitdp@\enit@meaning\enit@a}%
      {\edef\enit@c{\enit@c\enit@eltlevels{\enit@a}}}%
      {\edef\enit@b{\enit@b\enit@eltnames{\enit@a}}}}%
  \ifx\enit@b\@empty
     \def\enit@b{\enit@eltnames{list}}%
  \fi
  \ifx\enit@c\@empty
     \def\enit@c{\enit@eltlevels{0}}%
  \fi
  \def\enit@eltnames##1{%
    \def\enit@a{##1}%
    \enit@c}%
  \def\enit@eltlevels##1{%
    \enit@saveset\enit@a{##1}#1{#3}}%
  \enit@b}%

\let\c@enit@cnt\@tempcnta

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

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

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

\def\enit@ifunset#1{%
   \ifcsname#1\endcsname
     \expandafter\ifx\csname#1\endcsname\relax
       \enit@afterelse\expandafter\@firstoftwo
     \else
       \enit@afterfi\expandafter\@secondoftwo
     \fi
   \else
     \expandafter\@firstoftwo
   \fi
}

\def\enit@saveset#1#2#3#4{%
  \setcounter{enit@cnt}{#2}%
  \ifx\enit@forsize\@empty
    \ifcase#3%
      \expandafter
      \def\csname enit__block#1\romannumeral\c@enit@cnt\endcsname{#4}%
    \or
      \expandafter\let\expandafter\enit@b
        \csname enit__block#1\romannumeral\c@enit@cnt\endcsname
      \ifx\enit@b\relax
        \let\enit@b\@empty
      \fi
      \expandafter\def
        \csname enit__block#1\romannumeral\c@enit@cnt\expandafter\endcsname
        \expandafter{\enit@b,#4}%
```

```
    \fi
  \else
    \ifcase#3%
      \enit@ifunset{enit__block#1\romannumeral\c@enit@cnt}%
        {\expandafter\let
         \csname enit__block#1\romannumeral\c@enit@cnt\endcsname\@empty}%
        {}%
      \expandafter\let\expandafter\enit@b
        \csname enit__block#1\romannumeral\c@enit@cnt __blocksizes\endcsname
      \ifx\enit@b\relax
        \let\enit@b\@empty
      \fi
      \toks@\expandafter{\enit@b}%
      \edef\enit@b{\the\toks@\enit@forsize\enit@keys@sizes}%
      \expandafter\def
        \csname enit__block#1\romannumeral\c@enit@cnt __blocksizes\expandafter\endcsname
        \expandafter{\enit@b{#4}}%
    \else
      \enit@error{* and \string<\string> are not compatible}%
        {Use either * or angles, but not both.}%
    \fi
\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.

```
\let\enitdp@enumerate\@enumdepth
\let\enitdp@itemize\@itemdepth
\let\enitdp@description\@descriptiondepth
```

To put keys set with `\setlist` into the keyp 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.

```
\AddToHookWithArguments{blockenv}{%
```

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

```
  \enit@ifunset{enitdp@#1}{}%
  {%
```

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

```
    \@tempcnta\csname enitdp@#1\endcsname
    \advance\@tempcnta\@ne
    \enit@setkeys{list}%
    \enit@setkeys{list\romannumeral\@tempcnta}%
    \enit@setkeys{#1}%
    \enit@setkeys{#1\romannumeral\@tempcnta}%
  }%
}
```

```
\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.

```
\cs_new_protected:Npn \enit@setkeys #1 {
  \enit@ifunset {enit__block#1} {}
    {
      \clist_put_right:Nv \UnusedTemplateKeys { enit__block #1 }
    }
}
\makeatother
```

#### 7.3.11 \setlistdepth

#### 7.3.12 \AddEnumerateCounter

#### 7.3.13 \SetEnumitemSize

#### 7.3.14 \restartlist

⟨/package⟩