

# Customizing lists with the enumitem package

Javier Bezos\*

Version 3.8  
2019/02/04

## Contents

<b>1 Introduction</b>	<b>2</b>
<b>2 Quick reference</b>	<b>3</b>
<b>3 Keys</b>	<b>4</b>
3.1 Label and cross references format . . . . .	4
3.2 Horizontal spacing of labels . . . . .	6
3.3 Numbering, stopping, and resuming . . . . .	9
3.4 Series . . . . .	10
3.5 Penalties . . . . .	11
3.6 Injecting code . . . . .	11
3.7 Description styles . . . . .	12
3.8 Compact lists . . . . .	13
3.9 “Wide” lists . . . . .	13
3.10 enumerate-like labels . . . . .	13
3.11 Generic keys and values . . . . .	14
<b>4 Inline lists</b>	<b>14</b>
<b>5 Global settings</b>	<b>15</b>
<b>6 Size dependent settings</b>	<b>16</b>
<b>7 Cloning the basic lists</b>	<b>17</b>
<b>8 More about counters</b>	<b>18</b>
8.1 New counter representation . . . . .	18
8.2 Restarting enumerates . . . . .	19
<b>9 Package options</b>	<b>19</b>
<b>10 The trivlist issue</b>	<b>19</b>
<b>11 Samples</b>	<b>20</b>
<b>12 Afterword</b>	<b>22</b>
12.1 L <sup>A</sup> T <sub>E</sub> X lists . . . . .	22
12.2 Known issues . . . . .	23
12.3 What’s new in 3.0 . . . . .	23
12.4 Bug fixes . . . . .	23
12.5 Acknowledgements . . . . .	24

---

\*For bug reports, comments and suggestions go to <http://www.texnia.com/enumitem.html>.

**NOTE** Changes and new features with relation to version 3.5 are highlighted with `New X.X`. The most recent features could be still unstable. Please, report any issues you find on <https://github.com/jbezos/enumitem/issues>, which is better than just complaining on an e-mail list or a web forum. Forking and and pull requests are welcome.

**WARNING** Version 3.6 introduced two new keys: `left` and `first`. If your documents define some series with these names, an error is raised. Just rename them, or set the package option `series=override`.

## 1 Introduction

This package provides most of the flexibility you may want to customize the three basic list environments (`enumerate`, `itemize` and `description`) and to design your own lists, with a `<key>=<value>` syntax:

- Fancy labels and fancy refs, including a syntax similar to that in the `enumerate` package.
- Alternative ways for positioning the label, with a new parameter (`labelindent`) and a tab-like setting (`left`).
- Settings applied globally or only in one of the three types or even in a single list (including `\topsep`).
- Inline lists.
- Several description styles (which fix some bad spacing, too).
- `trivlists` properly formatted.
- Gathering of lists to be treated like a unit, as well as counter resuming.

In the interface a sort of “inheritance” is used. You can set globally the behavior of lists and then override several parameters of, say, `enumerate` and then in turn override a few parameters in a particular instance. The values will be searched in the hierarchy.

The package extends the syntax of the lists to allow an optional argument where a set of parameters in the form `key=value` are available. These keys are equivalent to the well known list parameters. Please, see a  $\text{\LaTeX}$  manual for a description of them. Next sections explains the extensions provided by `enumitem`.

- Vertical spacing:
  - `topsep`
  - `partopsep`
  - `parsep`
  - `itemsep`
- Horizontal spacing:
  - `leftmargin`
  - `rightmargin`
  - `listparindent`
  - `labelwidth`
  - `labelsep`
  - `itemindent`

**EXAMPLE** A straightforward example is:

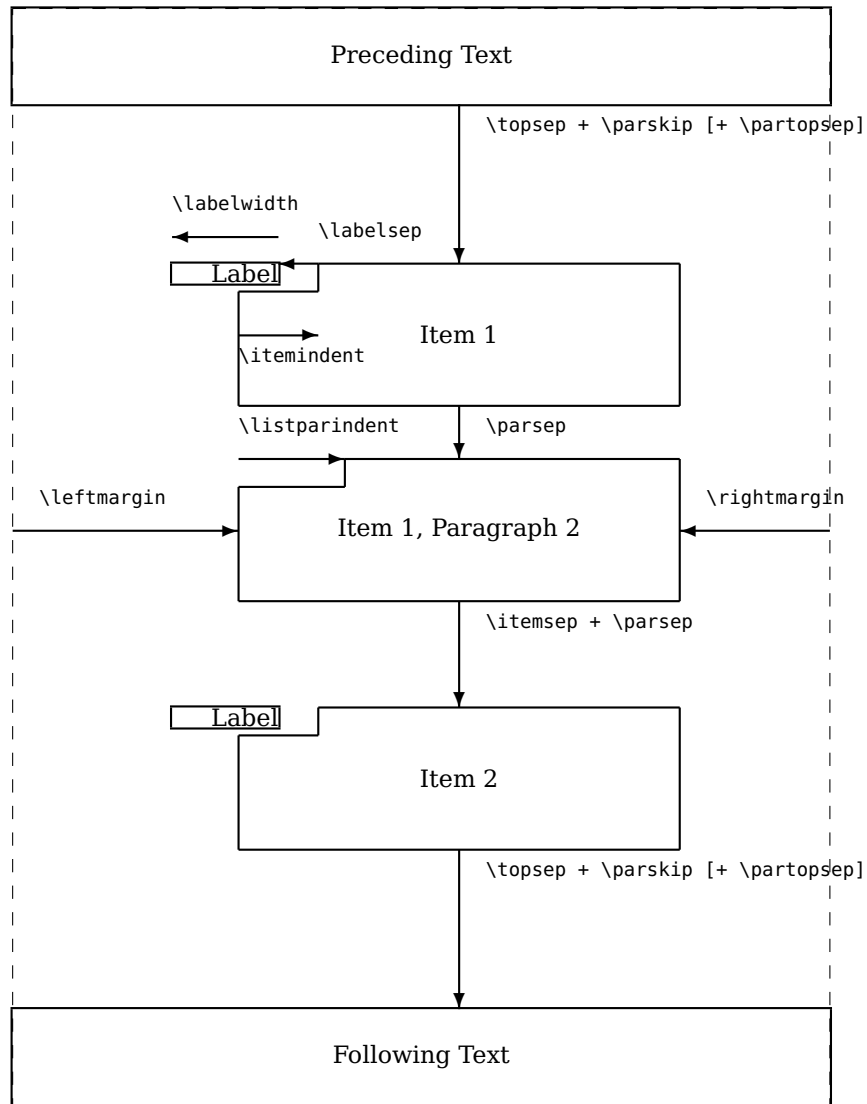


Figure 1: List parameters

```
\begin{itemize}[itemsep=1ex, leftmargin=1cm]
```

**NOTE** A way to see how these parameters work is with the `layouts` package (the manual is named `layman.pdf`). See figure 1.

## 2 Quick reference

Some common settings. See the manual below for details.

- To remove the vertical space altogether in a list:

```
\begin{enumerate}[nosep]
```

- To remove the vertical space altogether in *all* lists:

```
\setlist{nosep}
```

- To start the label at the margin and the item text at the current parindent:

```
\begin{enumerate}[left=0pt .. \parindent]
```

- To configure the labels like in enumerate: include the package option `shortlabels` and then, as a first element, write your label:

```
\begin{enumerate}[(1)]
```

- To continue the previous list, after a “pause”:

```
\begin{enumerate}[resume*]
```

- To use the three basic list in line: just add the package option `inline` and then the environments `enumerate*`, `itemize*` and `description*`.
- To set a numeric label with parenthesis, but a cross-reference without them:

```
\begin{enumerate}[label=(\arabic*), ref=\arabic*]
```

### 3 Keys

This section describes the keys in displayed lists. Most of them are available in inline lists, where further keys are available (see 4).

**WARNING** If the value is completely enclosed in braces, they are stripped off. If you want the braces, they must be duplicated. This is the default behavior of `keyval`, which `enumitem` just emulates.

#### 3.1 Label and cross references format

`label=<commands>`

Sets the label to be used in the current level. A set of starred versions of `\alph`, `\Alph`, `\arabic`, `\roman` and `\Roman`, without argument stand for the current counter in `enumerate`.<sup>1</sup> It works with `\value`, too (provided the widest label is not to be computed or `widest*` is used, see below).

**NOTE** If you prefer setting labels like the `enumerate` package, use “short labels” (see section 3.10).

**EXAMPLE** The following prints *a*), *b*), and so on (this is a standard style in Spanish, and formerly used by Chicago, too).

```
\begin{enumerate}[label=\emph{\alph*}]
```

**WARNING** The value of `label` is a moving argument, and fragile commands must be protected *except* the counters. Because of that, use of `\value` is somewhat tricky, because `\the` or `\ifnum` expects an actual value, which is not the case when `label` is being processed to replace internally the `*` by the form with the counter argument. The best solution is usually encapsulating the logic inside a new “counter” with the help of `\AddEnumerateCounter`.<sup>2</sup>

**EXAMPLE** A fancier example (which looks ugly, but it is intended only to illustrate what is possible; requires `color` and `pifont`):

---

<sup>1</sup>Actually, the asterisk is currently the argument but things may change. Consider them as starred variants and follow the corresponding syntax.

<sup>2</sup>Which is admittedly somewhat convoluted. A better way to accomplish this is on the way.

```
\begin{enumerate}[label=\protect\colorbox{blue}{yellow}\protect\ding{\value*}]
```

`label*=<commands>`

Like `label` but its value is appended to the parent label. For example, the following defines a legal list (1., 1.1., 1.1.1., and so on):

```
\newlist{legal}{enumerate}{10}  
\setlist{legal}{label*=\arabic*.
```

`ref=<commands>`

By default, label sets also the form of cross references and `\the...` (overriding the settings in parent hierarchical levels), but you can define a different format with this key. For example, to remove the right parenthesis:

```
\begin{enumerate}[label=\emph{\alph*}), ref=\emph{\alph*}]
```

**NOTE** In both `label` and `ref`, the counters can be used as usual. So, and provided the current level is the second one:

```
\begin{enumerate}[label=\theenumi.\arabic*.]
```

or

```
\begin{enumerate}[label=\arabic{enumi}.\arabic*.]
```

**NOTE** The labels are *not* accumulated to form the reference. If you want, say, something like 1.a from 1) as first level and a) as second level, you must set it with `ref`. You may use `\ref{level1}.\ref{level2}` with appropriate `ref` settings, but as Robin Fairbairns points out in the `TEX` FAQ:

... [that] would be both tedious and error-prone. What is more, it would be undesirable, since you would be constructing a visual representation which is inflexible (you could not change all the references to elements of a list at one fell swoop).

This is sensible and I recommend to follow the advice, but sometimes you might want something like:

```
... subitem \ref{level2} of item \ref{level1} ...
```

**WARNING** The value of `ref` is a moving argument, and fragile commands must be protected *except* the counters.

`font=<commands>`    `format=<commands>`

Sets the label font. Useful when the label is changed with the optional argument of `\item` and in description. The last command in `<commands>` can take an argument with the item label. In description class setting are in force, so you may want begin with `\normalfont`. A synonymous is `format`. Actually, this key may be used for any stuff to be executed at each `\item`, just before the label.

```
align=left align=right align=parleft
```

How the label is aligned (with relation to the label box edges). Three values are possible: `left`, the default `right` and `parleft` (a parbox of width `\labelwidth` with flush left text). The parameters controlling the label spacing should be properly set, either by hand or more conveniently with the `*` settings (see below):

```
\begin{enumerate}[label=\Roman*., align=left, leftmargin=*]
```

When the label box is supposed to have its natural width, use `left`.

```
\SetLabelAlign{<value>}{<commands>}
```

New align types can be defined (or the existing ones redefined) with `\SetLabelAlign`; the predefined values are equivalent to:

```
\SetLabelAlign{right}{\hs\llap{#1}}
\SetLabelAlign{left}{#1\hfil}
\SetLabelAlign{parleft}{\strut\smash{\parbox[t]{\labelwidth}{\raggedright#1}}}
```

**EXAMPLE** Although primarily intended for the alignment, this commands has other uses (an example is the provided `parleft`). For example, with the following all labels with `align=right` are set as superscripts:

```
\SetLabelAlign{right}{\hs\llap{\textsuperscript{#1}}}
```

A new name is also possible, of course.

**NOTE** If the last thing in the definition is a skip (typically `\hfil`), it is removed sometimes by description. If for some reason you want to avoid this, just add `\null` at the end.

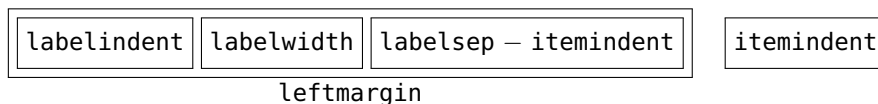
**NOTE** If you want the internal settings for `align` and `font` be ignored, you can override the `enumitem` definition of `\makelabel` in before:

```
\begin{description}[before={\renewcommand\makelabel[1]{\ref{##1}}}]
```

Alternatively, define a macro and use `\let`.

### 3.2 Horizontal spacing of labels

The horizontal space in the left margin of the current level is distributed in the following way:<sup>3</sup>



Here `labelindent` is a new parameter introduced by `enumitem`, described below. The rest are those in standard  $\text{\LaTeX}$ .

Actually, the layout is more complex because the label box (ie, `labelwidth`) could stick into the margin, which means `labelindent` takes a negative value.

**NOTE** Since `\parindent` is not used as such inside lists, but instead is set internally to either `\itemindent` or `\listparindent`, when used as the value of a parameter `enumitem` returns the global value, i. e., the value it has outside the outermost list.

<sup>3</sup>Admittedly, these figures are not exactly the clearest possible, and I intend to improve them in a future release.

**NOTE** **New 3.6** If you find these parameters baffling, you are not alone. You can visualize them by writing `\DrawEnumitemLabel` just before the first item (or in `first`), which draws 4 rules from top to bottom, `leftmargin` (thin), `labelindent` (thick if positive, thin if negative), `labelwidth` (thick) and `labelsep` (thick if positive, thin if negative). If 0, a rule is not shown. [Known issues: it seems it misplaces the rules with description and/or negative values.]

```
labelindent=<length>
\labelindent
```

This parameter is added in `enumitem` for the blank space from the margin of the enclosing list/text to the left edge of the label box. This means there is a redundancy because one of the parameters depends on the others, i.e., it has to be computed from the other values, as described below. By default, the computed value is `labelindent`, even if explicitly set with some value (it defaults to 0 pt). So, if you are setting it to some value, very likely you want to set some other parameter to `!` or `*`, because otherwise it is ignored.

There is a new counter length `\labelindent`.

The five parameters are related in the following way:

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

```
left=<labelindent>
left=<labelindent> .. <leftmargin>
```

**New 3.6** This is a convenience key to set quickly the most common layouts for the label. You may regard it as a sort of “rule” with two tab stops: the start of the label and the start of the text (both with relation to the normal side margin). With only `<labelindent>`, the left margin (the “start of text”) is computed with the `labelsep`. It internally resorts to `widest`, so the restrictions of the letter with relation to description also applies here: you might need change the computed parameter (eg, `itemindent=* with align=left`).

**EXAMPLE** Typical settings would be:

```
\begin{enumerate}[left= 0pt]
\begin{enumerate}[left= 0pt .. \parindent]
\begin{enumerate}[left= \parindent]
\begin{enumerate}[left= \parindent .. 2\parindent]
\begin{enumerate}[left= -\parindent .. 0pt]
```

**NOTE** The label width is set to the default widest one. If there are lists with Arabic numerals  $\geq 10$ , you may want to set `widest`, too.

**NOTE** `left=<labelindent>` sets `leftmargin=*`; `left=<labelindent> .. <leftmargin>` sets `labelsep=*`.

```
leftmargin=!   itemindent=!   labelsep=!   labelwidth=!   labelindent=!
```

Sets which value is to be computed from the others. The default is `labelindent=!`, but note some keys set another value (wide and description styles). Computations are done after *all* keys has been read. Explicit values are not lost, and so with the following hierarchical settings:

```
leftmargin=2em
labelindent=1em, leftmargin=!
labelindent=!
```

`leftmargin` is again 2em and `labelindent` is the computed parameter.

**NOTE** With `align=right` (the default), `labelindent=!` and `labelwidth=!` behave similarly in practice.

```
leftmargin=*   itemindent=*   labelsep=*   labelwidth=*   labelindent=*
```

Like before, but in addition `labelwidth` is first set to the width of the current label, using the default value of 0 in `\arabic*`, `viii` in `\roman*`, `m` in `\alph*` and similarly in uppercase forms (these values can be changed with `widest`, see below). Examples are:

```
\begin{itemize}[label=\textbullet, leftmargin=*]
\begin{enumerate}[label=\roman*], leftmargin=*, widest=iii]
\begin{itemize}[label      = \textbullet,
                  leftmargin = 2pc,
                  labelsep   = *]
\begin{enumerate}[label      = \arabic*.,
                  labelindent = \parindent,
                  leftmargin  = 2\parindent,
                  labelsep    = *]
```

The most useful are `labelsep=*` and `leftmargin=*`. With the former the item body begins at a fixed place (namely, `leftmargin`), while with the latter begins at a variable place depending on the label (but always the same within a list, of course).

**NOTE** Unfortunately, L<sup>A</sup>T<sub>E</sub>X does not define a default `labelsep` to be applied to all lists—simply the current value is used. With `enumitem` you can set default values for every list, as described below, and so, if you want to make sure `labelsep` is under your control, all you need is something like:

```
\setlist{labelsep=.5em}
```

**NOTE** `labelwidth=*` and `labelwidth=!` are synonymous.

```
widest=<string>   widest*=<integer>   widest
```

To be used in conjunction with the `*`-values, if desired. It overrides the default value for the widest printed counter. Sometimes, if lists are not very long, a value of `a` for `\alph` is more sensible than the default `m`:

```
\begin{enumerate}[leftmargin=*, widest=a] % Assume standard 2nd level
```

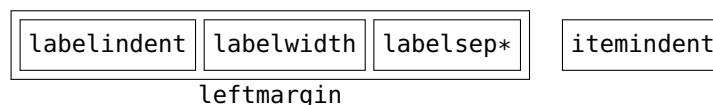
With no value, the default is restored. With `widest*`, the string is built using `<integer>` as the value of the counter (e.g., with `\roman`, `widest=viii` and `widest*=8` are the same).

Since `\value` does not return a string but a number, `widest` and the `*` values cannot be used with it. However, with `widest*`, being a number, it is allowed.

**New 3.6** It can be used with `itemize` and `description`, too. However, since the latter does some tricky formatting inside the label you might need change the computed parameter (eg, `itemindent=*` with `align=left`).

```
labelsep*=<length>
```

Remember `labelsep` spans part of `leftmargin` and `itemindent` if the latter is not zero. This is often somewhat confusing, so a new key is provided—with `labelsep*` the value is reckoned from the left margin (it just sets `\labelsep` and then adds `\itemindent` to it, but in addition later changes to `itemindent` are taken into account):



```
labelindent*=<length>
```

Like `labelindent`, but it is reckoned from the left margin in the current list and not from that in the enclosing list/text.



**EXAMPLE** A first pattern aligns the label with the surrounding `\parindent` while the item body is indented depending on the label and a fixed `labelsep`:

```
labelindent = \parindent,  
leftmargin  = *
```

A fairly frequent variant is aligning the label with the surrounding text (remember `labelindent` is `0pt` by default if it is not the computed parameter):

```
leftmargin = *
```

The former looks better in the first level while the latter seems preferable in subsequent ones. That can be easily set with

```
\setlist{leftmargin=*}  
\setlist[1]{labelindent=\parindent} % Only the level 1
```

**EXAMPLE** A second pattern aligns the item body with the surrounding `\parindent`. In this case (remember `labelindent` is the computed parameter if not set):

```
leftmargin = \parindent
```

**EXAMPLE** A third pattern would be the label aligned with `\parindent`, and the item body with `2\parindent`:

```
labelindent = \parindent,  
leftmargin  = 2\parindent,  
itemsep     = *
```

Again, a variant would be the label aligned with the surrounding text, and the item body with `\parindent`:

```
leftmargin = \parindent,  
itemsep    = *
```

### 3.3 Numbering, stopping, and resuming

`start=<integer>`

Sets the number of the first item.

`resume`

The counter continues from the previous `enumerate`, instead of being reset to 1.

```
\begin{enumerate}  
\item First item.  
\item Second item.  
\end{enumerate}  
Text.  
\begin{enumerate}[resume]  
\item Third item  
\end{enumerate}
```

This is done locally. If you want global resuming, see next section on series.

resume\*

Like resume but the options from the previous list are used, too. This option must be restricted to the optional argument in a environment (this is the only place where it makes sense). It should be used sparingly—if you are using it often, then very likely you want to define a new list (see 7). Further keys are allowed, and in this case the saved options are overridden by those in the current list (i.e., the position of resume\* does not matters). If there is a series of a certain list with resume\*, options are taken from the list previous to the first one, except for start.

**EXAMPLE** For example:

```
\begin{enumerate}[resume*, start=1] % or [start=1, resume*]
```

uses the keys in the previous enumerate, but restarts the counter.

### 3.4 Series

```
series=<series-name>  
<series-name> resume*=<series-name> resume=<series-name>
```

Another method to continue lists is by means of the key series, so that they behave like a unit. A list with key series is considered the starting list and its settings are stored *globally*, so that they can be used later with resume/resume\*. All these keys take a value with the series name (which must be different from existing keys):

- resume=<series-name> just continue numbering items in the series,
- resume\*=<series> also applies the settings of the starting list,
- <series>, i.e., the series name used as a key, is an alternative to resume\*=<series>.

**EXAMPLE** Consider:

```
\begin{enumerate}[label=\arabic*(a), leftmargin=1cm, series=l_after]  
  \item A  
  \item B  
\end{enumerate}
```

You get: 1(a) 2(a). You can continue with:

```
\begin{enumerate}[label=\arabic*(b), resume*=l_after]  
  % or [label=\arabic*(b), l_after]  
  \item A  
  \item B  
\end{enumerate}
```

You get: 3(b) 4(b). (But you can use start=1, if you like.)

Note you can add further arguments, which are executed after those saved at the starting list and therefore take precedence over them – in particular, resume\* itself takes precedence over a start (e.g., start=1) in the the starting list.

**NOTE** Every time a series is started, several commands are defined internally. Thus, to avoid wasting resources use the same name for non-overlapping series.

**WARNING** The package may introduce new keys in the future, so using directly `<series>` as a key is a potential source of forward incompatibilities. However, it's safe using a non-letter character other than hyphen or star in the key name (e.g., `!notes` or `m_steps`), as well as uppercase letters and digits, because `enumitem` will never use them. **New 3.7** If you have defined some series with an all lowercase name and a new conflicting key has been introduced, an alternative to changing their names is the package option `series=override` (the error message is `Invalid series name 'key'`). With it series names take precedence over predefined keys – but use it only when absolutely necessary.

### 3.5 Penalties

```
beginpenalty=<integer> midpenalty=<integer> endpenalty=<integer>
```

Set the penalty at the beginning of a list, between items and at the end of the list, respectively. Please, refer to your L<sup>A</sup>T<sub>E</sub>X or T<sub>E</sub>X manual about how penalties control page breaks. Unlike other parameters, when a list starts their values are not reset to the default, thus they apply to the child lists.

### 3.6 Injecting code

```
before=<code> before*=<code>
```

Execute code before the list starts (more precisely, in the second argument of the `list` environment used to define them). The unstarred form sets the code to be executed, overriding any previous value, while the starred one adds the code to the existing one (in the setting hierarchy, see below, *not* with relation to the enclosing list/text). It can contain, say, rules and text, but this has not been extensively tested. All calculations have been finished, and you can access and manipulate the list parameters.

**EXAMPLE** To have both margins (left and right) set to the widest label:

```
\setlist[leftmargin=*, before=\setlength{\rightmargin}{\leftmargin}]
```

```
after=<code> after*=<code>
```

Same, but just before the list ends.

```
first=<code> first*=<code>
```

**New 3.6** Same, but as the very first thing in the list body, so that

```
\begin{itemize}[first=<code>]
```

is the same as

```
\begin{itemize}
<code>
```

**EXAMPLE** With `first` you can define your own environments for displayed material. A trivial example is:

```
\newlist{letter}{itemize}{1}
\setlist[letter]{first=\item[]\itshape, rightmargin=\leftmargin}
```

Here there is no need for a `label`, because it is not used.

`\EnumitemId`

**New 3.7** To help in some tasks, a unique numeric identifier is assigned to each list, returned by `\EnumitemId`.

**EXAMPLE** Here is an example of how to combine a `\label` with `\EnumitemId`, and after to automatically set the width of the list label to the widest one (provided the ref is the same as the label):<sup>4</sup>

```
\SetEnumitemKey{widestlabel}
{labelwidth = \widthof{\ref{enum-\EnumitemId}},
after      = \label{enum-\EnumitemId}}
```

Then just use the key `widestlabel`.

**EXAMPLE** Reverse counting is also doable, but somewhat trickier, and we need some “external” help. Here is a possible solution, but not the only one (and very likely not even the best – for example, `start` is in fact no-op).

```
\usepackage{calc,cleveref,crossreftools}
\crtrefundefinedtext{0}

\newcounter{revcount}
\newcommand\revcounter[1]{%
  \setcounter{revcount}{1+\crtcrefnumber{enum-\EnumitemId}-\value{#1}}
\AddEnumerateCounter\revcounter\revcounter{} % the 2nd is dummy

\SetEnumitemKey{revarabic}
{label = \revcounter*(\arabic{revcount}),
ref    = (\arabic{revcount}),
after  = \label{enum-\EnumitemId}}
```

Note `ref` must be set separately, because `\revcounter` is not expandable.

### 3.7 Description styles

A key available in description.

`style=<name>`

Sets the description *style*. `<name>` can be any of the following:

**standard** Like description in standard classes, although with other classes it could be somewhat different. The label is boxed. Sets `itemindent=!`.

**unboxed** Much like the standard description, but the label is not boxed to avoid uneven spacing and unbroken labels if they are long. Sets `itemindent=!`.

**nextline** If the label does not fit in the margin, the text continues in the next line, otherwise it is placed in a box of width `\leftmargin - \labelsep`, i.e., the item body never sticks into the left margin. Sets `labelwidth=!`.

**same** Like `nextline` but if the label does not fit in the margin the text continues in the same line. Same as `style=unboxed,labelwidth=!`.

**multiline** The label is placed in a parbox whose width is `leftmargin`, with several lines if necessary. Same as `style=standard,align=parleft,labelwidth=!`.

**WARNING** 1. Mixing boxed and unboxed labels has not a well-defined behavior.

2. When nesting list all combinations are allowed but not all make sense.

3. Nesting `nextline` lists is not supported (it works, but its behavior might change in the future, because the current one is not what one could expect).

<sup>4</sup>See <https://tex.stackexchange.com/questions/29322/how-to-make-enumerate-items-align-at-left-margin>.

### 3.8 Compact lists

```
noitemsep    nosep
```

The key `noitemsep` kills the space between items and paragraphs (i.e., `itemsep=0pt` and `parsep=0pt`), while `nosep` kills all vertical spacing.<sup>5</sup>

### 3.9 “Wide” lists

```
wide  
wide=<parindent>
```

With this convenience key, the `leftmargin` is null and the label is part of the text—in other words, the items look like ordinary paragraphs.<sup>6</sup> Here `labelsep` sets the separation between the label and the first word. It is equivalent to

```
align=left, leftmargin=0pt, labelindent=\parindent,  
listparindent=\parindent, labelwidth=0pt, itemindent=!
```

With `wide=<parindent>` you may set at once another value instead of `\parindent`. Of course, these keys can be overridden after `wide`, too; for example, remembering that with left-aligned labels the text is pushed if they are wider than `labelwidth`, you can set `labelwidth=1.5em` for a minimal width, or instead of `itemindent=!` you may prefer `itemindent=*`, which sets the minimal width to that of widest label. In level 2 you may prefer `labelindent=2\parindent`, and so on. You may also want to combine it with `noitemsep` or `nosep`.

### 3.10 enumerate-like labels

```
shortlabels (package option)
```

With the package option `shortlabels` you can use an enumerate-like syntax, where `A`, `a`, `I`, `i` and `1` stand for `\Alph*`, `\alph*`, `\Roman*`, `\roman*` and `\arabic*`. This is intended mainly as a sort of compatibility mode with the `enumerate` package, and therefore the following special rule applies: if the very first option (at any level) is not recognized as a valid key, then it will be considered a label with the enumerate-like syntax. For example:

```
\begin{enumerate}[i], labelindent=\parindent, labelsep=*]  
...  
\end{enumerate}
```

You may want to set `ref`, too, if different from the label.

Although perhaps not so useful, you can omit `label=` in the `itemize` environment under similar conditions, too:

```
\begin{itemize}[\textbullet]  
...  
\end{itemize}
```

```
\SetEnumerateShortLabel{<key>}{<replacement>}
```

With this command, you can define new keys (or redefine them), which is particularly useful for `enumerate` to be adapted to specific typographical rules or to extend it for non-Latin scripts. Here `<replacement>` contains one of the starred versions of counters.

**EXAMPLE** For example:

<sup>5</sup>The key `nolistsep`, now deprecated, introduced a thin stretch, which was not the intended behavior.

<sup>6</sup>`fullwidth` is deprecated.

```
\SetEnumerateShortLabel{i}{\textsc{\roman*}}
```

redefines `i` so that items using this key are numbered with small caps roman numerals.

**NOTE** The key has to be a single character.

### 3.11 Generic keys and values

```
\SetEnumitemKey{<key>}{<replacement>}
```

With this command you can create your own (valueless) keys. Keys so defined can then be used like the others.

**EXAMPLE** With

```
\SetEnumitemKey{midsep}{topsep=3pt, partopsep=0pt}
```

you may write

```
\begin{enumerate}[midsep]
```

**EXAMPLE** Another example is multicolumn lists, with `multicol`:

```
\SetEnumitemKey{twocol}{
  itemsep = 1\itemsep,
  parsep  = 1\parsep,
  before  = \raggedcolumns\begin{multicols}{2},
  after   = \end{multicols}}
```

Here, the settings for `itemsep` and `parsep` kill the stretch and shrink parts, which in this case improves the result. Of course, you may want to define a new list.

**WARNING** The package may introduce new keys in the future, so `\SetEnumitemKey` is a potential source of forward incompatibilities. However, it's safe using a non-letter character other than hyphen or star in the key name (e.g., `:name` or `2_col`), as well as uppercase letters and digits, because `enumitem` will never use them.

```
\SetEnumitemValue{<key>}{<string-value>}{<replacement>}
```

This command provides a further abstraction layer for the `<key>=<value>` pairs. With it you can define logical names which are translated to the actual value. For example, with:

```
\SetEnumitemValue{label}{numeric}{\arabic*}
\SetEnumitemValue{leftmargin}{standard}{\parindent}
```

you might say:

```
\begin{enumerate}[label=numeric, leftmargin=standard]
```

So, you can leave to the final design what `label=numeric` means.

## 4 Inline lists

Inline lists are “horizontal” lists set as ordinary text inside a paragraph. With this package you can create inline lists, as explained below, with `\newlist`, which have their own labels and counters. However, very often inline versions of standard lists, with the same labeling schema, will be enough – the package option `inline` does that.

**WARNING** Items are boxed, so floats are lost and nested lists are not allowed (remember many displayed elements are defined as lists). Display math is forbidden too, and due to an optimization done by T<sub>E</sub>X when building lists, explicit hyphenation may be wrong.<sup>7</sup> There was a reason for this default setting, namely, this feature was mainly devised for short items (a few words), and the parameter `itemjoin*` could be useful for logical markup. To overcome these limitations, you may set `mode=unboxed`, described below.

<code>inline</code> (package option)
<code>enumerate*</code> <code>itemize*</code> <code>description*</code> (environments)

With the package option `inline`, three environments for inline lists are defined: `enumerate*`, `itemize*`, and `description*`. They emulate the behavior of `paralist` and `shortlist` in that labels and settings are shared with the displayed (ie, “normal”) lists `enumerate`, `itemize` and `description`, respectively (however, remember resuming is based on environment names, not on list types). This applies only to those created with `inline` – inline lists created with `\newlist` as described below are independent and use their own labels and settings.

**NOTE** Note `inline` is not required if you do not need the inline versions of standard lists, but instead you define your own standalone inline lists with `\newlist`.

**WARNING** Settings for these three environments as defined by `inline` are shared with the displayed variants, so they cannot be redefined directly with `\newlist`. Trying to do it raises a cryptic error. If you need separate setting, define them with `\newlist` and not with `inline`.

<code>itemjoin=&lt;string&gt;</code> <code>itemjoin*=&lt;string&gt;</code> <code>afterlabel=&lt;string&gt;</code>
---

Format is set with keys `itemjoin` (default is a space), and `afterlabel` (default is `\nobreakspace`, ie, `~`). An additional key is `itemjoin*`, which, if set, is used instead of `itemjoin` before the last item.

`itemjoin` is ignored in vertical mode (i.e., in mode `unboxed` and just after a quote, a displayed list and the like).

**EXAMPLE** With

```
before=\unskip{ }, itemjoin={{; }}, itemjoin*={{, and }}
```

the following punctuation between items is used:

Blah blah: (a) one; (b) two; (c) three, and (d) four. Blah blah

<code>mode=unboxed</code> <code>mode=boxed</code>
---

If using floats, lists or displayed math inside inline lists is important, use an alternative “mode”, which you can activate with `mode=unboxed` (the default is `mode=boxed`). With it, floats may be used freely, but misplaced `\items` are not caught and `itemjoin*` is ignored (a warning is written to the log about this fact).

## 5 Global settings

Global changes, to be applied to all of these list, are also possible:

---

<sup>7</sup>A Knuthian “premature optimization”? Who knows, but anyway LuaT<sub>E</sub>X has removed it, so hyphenation with this engine should be correct.

```
\setlist[enumerate,⟨levels⟩]{⟨format⟩}
\setlist[itemize,⟨levels⟩]{⟨format⟩}
\setlist[description,⟨levels⟩]{⟨format⟩}
\setlist[⟨levels⟩]{⟨format⟩}
```

Where  $\langle level \rangle$  is the list level (one or more) in `list`, and the corresponding levels in `enumerate` and `itemize`.<sup>8</sup> With no  $\langle levels \rangle$ , the format applies to all of them. Here ‘list’ does not mean any list but only the three ones handled by this package, and those redefined by this package or defined with `\newlist` (see below). For example:

```
\setlist{noitemsep}
\setlist[1]{labelindent=\parindent} % < Usually a good idea
\setlist[itemize]{leftmargin=*}
\setlist[itemize,1]{label=$\triangleleft$}
\setlist[enumerate]{labelsep=*, leftmargin=1.5pc}
\setlist[enumerate,1]{label = \arabic*. ,
                      ref   = \arabic*}
\setlist[enumerate,2]{label = \emph{\alph*}},
                      ref   = \theenumi.\emph{\alph*}}
\setlist[enumerate,3]{label = \roman*},
                      ref   = \theenumii.\roman*}
\setlist[description]{font=\sfamily\bfseries}
```

These settings are read in the following order: `list`, `list` at the current level, `enumerate/itemize/description`, and `enumerate/itemize/description` at the current level; if a key appears several times with different values, the last one, i.e., the most specific one, is applied. If we are resuming a series or a list with `resume*`, the saved keys are then applied. Finally, the optional argument (except `resume*`), if any, is applied.

L<sup>A</sup>T<sub>E</sub>X provides a set of macros to change many of these parameters, but setting them with the package is more consistent and sometimes more flexible at the cost of being more “explicit” (and verbose).

The list specification can contain variables and counters, provided they are expandable, and counters are calc-savvy, so that if you load this package you can write things like:

```
\newcount{toplist}
\setcount{toplist}{1}
\newcommand{\mylistname}{enumerate}
\setlist[\mylistname,\value{toplist}+1]{labelsep=\itemindent+2em}
```

This allows defining lists with the help of loops.

**WARNING** It seems there is no way to catch a misspelled name in `\setlist` or `\newlist`, and a meaningless error “Missing number, treated as zero” is raised.

## 6 Size dependent settings

**New 3.6** For settings depending on the font size, in most cases all you need are relative units like `ex` or `em`. Sometimes, you may want discrete steps, and a special syntax allows them.

The following package option is required for making use of this feature.

```
sizes (package option)
```

Lengths can contain size-based settings as follows (the value before the first `<` is a default).

```
\setlist{
  topsep      = 20pt <-10> 6pt          <10-> 40pt,
  leftmargin  = <-10> 0cm <10> 1cm <10-> 2cm ,
  rightmargin = <-10> 0cm <10> 1cm <10-> 2cm ,
}
```

<sup>8</sup>`\setenumerate`, `\setitemize` and `\setdescription` are deprecated.



Names are accepted, too: `script`, `tiny`, `footnote`, `small`, `normal`, `large`, `Large`, `LARGE`, `huge`, `Huge` (ie, remove ‘size’ from the L<sup>A</sup>T<sub>E</sub>X name if necessary). For example:

```
\setlist{
  topsep      = 20pt <-normal> 6pt          <normal-> 40pt,
  leftmargin  = <-normal> 0cm <normal> 1cm <normal-> 2cm ,
  rightmargin = <-normal> 0cm <normal> 1cm <normal-> 2cm ,
}
```

Single values take precedence over ranges (i. e., specific takes precedence over generic). In ranges, the last match wins. The range `a-b` is  $a \leq \text{size} < b$  (the lower bound is included, but not the upper one). These rules allow in the examples above the setting for `10` or `normal` in the logical place. Remember nominal sizes are not always the real sizes – for example, with option `11pt`, `\normalsize` (and `normal`) is 10.95. You may precede a value with several single qualifiers like `<small><normal>12pt`.<sup>9</sup> A value before the first `<. .>` is considered a default value.

**NOTE** For efficiency reasons, named sizes are assigned only once, when `enumitem` is loaded, in the assumption they are set by the class, or a local style loaded previously.

```
\SetEnumitemSize{<name>}{<selector>}
```

**New 3.7** If sizes are modified after loading `enumitem` or you are using a class with non standard sizes (or even you just want another names), they can be set or reset with the following tool.

**EXAMPLE** A trivial example:

```
\SetEnumitemSize{normal}{\normalsize}
\SetEnumitemSize{large}{\large}
```

```
\setlist<<size>>[<names>,<levels>]{<keys/values>}
```

**New 3.7** An extension to `\setlist` described below which adds the definitions, but only for the given size (either single or a range). The precedence rules for sizes also apply here (so that the order of `\setlist`’s are relevant), and size dependent keys as defined by this procedure take precedence over the rest of the keys. For example:

```
\setlist<-normal>[enumerate]{nosep}
```

However, only a size qualifier is accepted in each `\setlist`.

## 7 Cloning the basic lists

```
\newlist{<name>}{<type>}{<max-depth>}
\renewlist{<name>}{<type>}{<max-depth>}
```

The three lists can be cloned so that you can define “logical” environments behaving like them. To define a new lists (or redefine a existing one), use `\newlist` (or `\renewlist`), where `<type>` is `enumerate`, `itemize` or `description`. Inline lists have types `enumerate*`, `itemize*`, and `description*`.

**NOTE** The inline `<type>`s are available always, even without the package option `inline`, which just defines three environments of the corresponding types with those names.

<sup>9</sup>Note this syntax follows closely that of `\DeclareFontShape`, except in the precedence of single values.

If  $\langle type \rangle$  is `enumerate`, a set of counters with names  $\langle name \rangle i$ ,  $\langle name \rangle ii$ ,  $\langle name \rangle iii$ ,  $\langle name \rangle iv$ , etc. (depending on  $\langle max-depth \rangle$ ) is defined.

Then you can use those counters in labels; e. g., if you have defined a list named `steps`, you can define a label with:

```
label=\arabic{stepsii}.\arabic{stepsi}
```

**WARNING** Don't use an arbitrarily large number for  $\langle max-depth \rangle$ , to avoid creating too many counters and related macros.

**WARNING** For consistency with the counter naming schema in L<sup>A</sup>T<sub>E</sub>X, list levels are also named internally with a roman numeral, ie,  $\langle list \rangle i$ ,  $\langle list \rangle ii$ ,  $\langle list \rangle iii$ , etc. For this reason (both counter and list names), defining two lists as, say, `books` and `booksi` leads to unexpected results (currently without any warning, which should be fixed).

```
\setlist[ $\langle names \rangle$ , $\langle levels \rangle$ ]{ $\langle keys/values \rangle$ }
\setlist*[ $\langle names \rangle$ , $\langle levels \rangle$ ]{ $\langle keys/values \rangle$ }
```

After creating a list, you can (in fact you must, at least the label) set the new list with `\setlist`:

```
\newlist{ingredients}{itemize}{1}
\setlist[ingredients]{label=\textbullet}
\newlist{steps}{enumerate}{2}
\setlist[steps,1,2]{label=(\arabic*)}
```

Names in the optional argument of `\setlist` say which lists applies the settings to, and numbers say the level (it is calc-savvy). Several lists and/or several levels can be given, and all combinations are set; e.g.:

```
\setlist[enumerate,itemize,2,3]{...}
```

sets `enumerate/2`, `enumerate/3`, `itemize/2` and `itemize/3`. No number (or 0) means "all levels" and no name means "all lists"; no optional argument means "all lists at all levels".

The starred form `\setlist*` adds the settings to the previous ones. You may restrict the additions to a certain font size, as explained in section 6. It must be noted the latter have a higher precedence than the starred ones (i. e., settings added for some sizes take precedence over settings added for all sizes, so that the most specific value for a key is applied).

```
\setlistdepth{ $\langle integer \rangle$ }
```

By default, L<sup>A</sup>T<sub>E</sub>X has a limit of 5 nesting levels, but when cloning lists this value may be too short, and therefore you may want to set a new value. In levels below the 5th (or the deepest defined by a class), the settings of the last are used (i.e., `\@listvi`).

## 8 More about counters

### 8.1 New counter representation

```
\AddEnumerateCounter{ $\langle LaTeX command \rangle$ }{ $\langle internal command \rangle$ }{ $\langle widest label \rangle$ }
```

"Registers" a counter representation so that `enumitem` recognizes it. Intended mainly for non Latin scripts, but also useful in Latin scripts.

**EXAMPLE** The following example defines a new counter with named ordinals:

```

\makeatletter
\def\c@text#1{\expandafter\@c@text\csname c@#1\endcsname}
\def\@c@text#1{\ifcase#1\or First\or Second\or Third\or
Fourth\or Fifth\or Sixth\fi}
\makeatother
\AddEnumerateCounter{\c@text}{\@c@text}{Second}

```

**NOTE** The counter names can contain @ even if not a letter without raising an error, as shown in the example above.

A starred variant allows to give a number instead of a string as the widest label.

**EXAMPLE** If the widest label is that corresponding to the value 2:

```

\AddEnumerateCounter*{\c@text}{\@ctmoreext}{2}

```

This variant is to be preferred if the representation is not a plain string but it is styled, e.g., with small caps.

**EXAMPLE** An example for Russian is:

```

\AddEnumerateCounter*{\@asbuk}{\c@asbuk}{7}

```

## 8.2 Restarting enumerates

```
\restartlist{<list-name>}
```

Currently you can get a continuous numbering through a document with:

```
\setlist[enumerate]{resume}
```

`\restartlist` has been added for restarting the counter in the middle of the document. For example, you could emit a `\restartlist` when chapters start, so that there is a continuous numbering through every chapter.

**WARNING** It is based solely in the list *name*, *not* the list *type*, which means `enumerate*` as defined with the package option `inline` is not the same as `enumerate`, because its name is different.

## 9 Package options

Besides `inline`, `ignoredisplayed`, `sizes`, `series=override` and `shortlabels`, the following option is available.

```
loadonly
```

With this package option the package is loaded but the three lists are not redefined. You can create your own lists, yet, or even redefine the existing ones.

## 10 The trivlist issue

L<sup>A</sup>T<sub>E</sub>X uses a simplified version of `list` named `trivlist` to set displayed material, like `center`, `tabbing`, `theorem`, etc., even if conceptually they are not lists. Unfortunately, `trivlist` uses the current list settings, which has the odd side effect that changing the vertical spacing of lists also changes sometimes the spacing in these environments.

This package modifies `trivlist` so that the default settings for the current level (ie, those set by the corresponding `clo` files) are set again. In standard  $\LaTeX$  that is usually redundant, but if we want to fine tune lists, not resetting the default values could be a real issue (particularly if you use the `nosep` option).

A minimal control of vertical spacing has been made possible with<sup>10</sup>

- `\setlist[trivlist,<level>]{<keys/values>}`

but `trivlist` itself, which is not used directly very often, does not accept an optional argument. This feature is not intended as a full-fledge `trivlist` formatter.

If for some reason you do not want to change `trivlist` and preserve the original definition, you can use the package option `ignoredisplayed`.

**New 3.6** If, on the other hand, you want to also apply the changes for all lists to `trivlists`, just set the package option `includedisplayed`.

## 11 Samples

In these samples we set `\setlist{noitemsep}`

---

```

En un lugar de la Mancha, de cuyo nombre no quiero acordarme,
no ha mucho tiempo que viv\{'i}a un hidalgo de los de
\begin{enumerate}[labelindent=\parindent,leftmargin=*]
  \item lanza en astillero,
  \item adarna antigua,
  \item roc\{'i}n flaco, y
  \item galgo corredor.
\end{enumerate}
Una olla de algo m\{'a}s vaca que carnero, salpic\{'o}n las m\{'a}s
noches, duelos y quebrantos los s\{'a}bados...
```

The rule shows `labelindent`.

---

```

En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un
hidalgo de los de
1. lanza en astillero,
2. adarna antigua,
3. rocín flaco, y
4. galgo corredor.
```

Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados...

---

With `\begin{enumerate}[leftmargin=*]` % `labelindent=0pt` by default.  
The rule shows `labelindent`.

---

```

En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un
hidalgo de los de
1. lanza en astillero,
2. adarna antigua,
3. rocín flaco, y
4. galgo corredor.
```

Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados...

---

With `\begin{enumerate}[leftmargin=\parindent]`.  
The rule shows `leftmargin`.

---

```

En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un
hidalgo de los de
1. lanza en astillero,
2. adarna antigua,
3. rocín flaco, y
```

---

<sup>10</sup>`\setdisplayed` is deprecated.

#### 4. galgo corredor.

Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados...

---

With `\begin{enumerate}[labelindent=\parindent, leftmargin=*, label=\Roman*., widest=III, align=left]`.

The rule shows `labelindent`. Note

---

En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de

- I. lanza en astillero,
- II. adarna antigua,
- III. rocín flaco, y
- IV. galgo corredor.

Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados...

---

With `\begin{enumerate}[label=\fbox{\arabic*}]`. A reference to the first item is 1

---

En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de

- 1 lanza en astillero,
- 2 adarna antigua,
- 3 rocín flaco, y
- 4 galgo corredor.

Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados...

---

With nested lists.

---

```
En un lugar de la Mancha, de cuyo nombre no quiero acordarme,
no ha mucho tiempo que viv\ '{i}a un hidalgo de los de
\begin{enumerate}[label=(\alph*), labelindent=\parindent,
  leftmargin=*, start=12]
  \item lanza en astillero,
  \begin{enumerate}[label=(\alph{enumi}.\roman*), leftmargin=*, start=7]
    \item adarna antigua,
  \end{enumerate}
  \item roc\ '{i}n flaco, y
  \begin{enumerate}[label=(\alph{enumi}.\roman*), leftmargin=*, resume]
    \item galgo corredor.
  \end{enumerate}
\end{enumerate}
Una olla de algo m\ '{a}s vaca que carnero, salpic\ '{o}n las m\ '{a}s
noches, duelos y quebrantos los s\ '{a}bados...
```

En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de

- (l) lanza en astillero,
  - (l.vii) adarna antigua,
- (m) rocín flaco, y
  - (m.viii) galgo corredor.

Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados...

---

```
En un lugar de la Mancha, de cuyo nombre no quiero acordarme,
no ha mucho tiempo que viv\ '{i}a un hidalgo de los de
\begin{description}[font=\sffamily\bfseries, leftmargin=3cm,
  style=nextline]
  \item[Lo primero que ten\ '{i}a el Quijote] lanza en astillero,
  \item[Lo segundo] adarna antigua,
  \item[Lo tercero] roc\ '{i}n flaco, y
```

```

\item[Y por \'{u}ltimo, lo cuarto] galgo corredor.
\end{description}
Una olla de algo m\{a}s vaca que carnero, salpic\{o}n las m\{a}s
noches, duelos y quebrantos los s\{a}bados...

```

---

En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de

**Lo primero que tenía el Quijote** lanza en astillero,  
**Lo segundo** adarna antigua,  
**Lo tercero** rocín flaco, y  
**Y por último, lo cuarto** galgo corredor.

Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados...

---

Same, but with sameline.

---

En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de

**Lo primero que tenía el Quijote** lanza en astillero,  
**Lo segundo** adarna antigua,  
**Lo tercero** rocín flaco, y  
**Y por último, lo cuarto** galgo corredor.

Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados...

---

Same, but with multiline. Note the text overlaps if the item body is too short.

---

En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de

**Lo primero que** lanza en astillero,  
**Lo segundo** adarna antigua,  
**Lo tercero** rocín flaco, y  
**Y por último, lo** galgo corredor.

**cuarto** Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados...

## 12 Afterword

### 12.1 L<sup>A</sup>T<sub>E</sub>X lists

As it is well known, L<sup>A</sup>T<sub>E</sub>X predefines three lists: `enumerate`, `itemize` and `description`. This is a very frequent classification which can also be found in, say, HTML. However, there is a more general model based in three fields—namely, label, title, and body—, so that `enumerate` and `itemize` has label (numbered and unnumbered) but no title, while `description` has title but no label. In this model, one can have a description with entries marked with labels, as for example (of course, this simple solution is far from satisfactory):

```

\newcommand\litem[1]{\item{\bfseries #1,\enspace}}
\begin{itemize}[label=\textbullet]
\litem{Lo primero que ten\{i}a el Quijote} lanza en astillero,
... etc.

```

---

En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de

- **Lo primero que tenía el Quijote,** lanza en astillero,
- **Lo segundo,** adarna antigua,
- **Lo tercero,** rocín flaco, y
- **Y por último, lo cuarto,** galgo corredor.

---

## 12.2 Known issues

- List resuming is based on environment names, and when a `\newenvironment` contains a list you may want to use `\begin{<list>}` and `\end{<list>}`. Using the corresponding low-level commands (ie, `\<list>` and `\end{<list>}`) is not an error, but it is your responsibility to make sure the result is correct.
- The behavior of mixed boxed labels (including `enumerate` and `itemize`) and unboxed labels is not well-defined. The same applies to boxed and unboxed inline lists (which could even raise an error).
- Similarly, resuming a series and a list at the same time is allowed, too, but again its behavior is not well-defined.
- (3.5.2) An incompatibility with 2.x has popped up – if you were using the optional argument to pass a value to a `\ref` or other macro requiring expandable macros, an error is raised. A quick fix is letting `\makelabel` to `\descriptionlabel` in before.

## 12.3 What's new in 3.0

- Inline lists, with keys to set how items are joined (ie, the punctuation between items). Two modes are provided: `boxed` and `unboxed`.
- `\setlist` is calc-savvy (eg, for use in loops), and you can set different lists and levels at once.
- All lengths related to labels can take the value `*` (and not only `labelsep` and `leftmargin`). Its behavior has been made consistent and there is new value `!` which does not compute the widest label.
- With `\restartlist{<list-name>}`, list counters can be restarted (in case you are using `resume`).
- `resume*` can be combined with other keys.
- Lists can be gathered globally using `series`, so that they are considered a single list. To start a series just use `series=<series-name>` and then resume it with `resume=<series-name>` or `resume*=<series-name>`.
- The “experimental” `fullwidth` has been replaced by a new key `wide`.
- `\SetLabelAlign` defines new align values.
- You can define “abstract” values (eg, `label=numeric`) and new keys.
- (3.2) `start` and `widest*` are calc-savvy.
- (3.2) `\value` can be used with `widest*`.
- (3.2) Some internal restrictions in `\arabic` and the like has been removed. It is more flexible at the cost of having a more “relaxed” error checking.

## 12.4 Bug fixes

- Star values (eg, `leftmargin=*`) could not be overridden and new values were ignored.
- `nolistsep` as the first of several keys was not always recognized and therefore treated like a short label (i.e., `no\roman*stsep`).
- `labelwidth` did not always work (when there was a prior `widest` and `*`)
- With `align=right` the label and the following text could overlap.
- `description` did not get the correct list level.
- At some point (2.x?) `\value*` stopped working.
- (3.1) Unfortunately, `xkeyval` “kills” `keyval`, so the latter has been replicated in `enumitem`.
- (3.3) Fixes a serious bug – with `*` neither `itemize` nor `description` worked.
- (3.4) Fixes bad spacing in mode `boxed` (misplaced `\unskip` before the first item and wrong `spacefactor` between items).
- (3.4) `nolistsep` did not work as intended, but since the error has been there for several years, a new key `nosep` is provided.
- (3.4) The issue with `nolistsep` with `shortlabels` (see above) was not fixed in all cases. Hopefully now it is.
- (3.5.0) Fixed the fix related to the `spacefactor` between items.
- (3.5.0) Fixed a problem with nested boxed inline lists.

- (3.5.1) `resume*` only worked once, and subsequent ones behaved like `resume`.
- (3.5.2) Fixed `\setlist*`, which didn't work.

## **12.5 Acknowledgements**

I wish to thank particularly the comments and suggestions from Lars Madsen, who has found some bugs, too.