Le package enumerate*

David Carlisle
carlisle@cs.man.ac.uk

Traduction française par Jean-Pierre Drucbert†

1999/03/05

Résumé

Ce package ajoute un argument optionnel à l'environnement enumerate pour déterminer la manière dont le compteur d'énumération doit être imprimé.

Une occurrence de l'un des caractères `A a I i` ou `1` produit la valeur du compteur imprimée avec (respectivement) `\Alph`, `\alph`, `\Roman`, `\roman` ou `\arabic`.

Ces lettres peuvent être entourées de chaînes quelconques mettant en jeu toute autre expression TEX, mais les caractères `A a I i 1` doivent être à l'intérieur d'un groupe `{ }` s'ils ne doivent pas avoir leur sens spécial.

1 Exemples

EX i. un un un un un un un un un un un un un un

EX ii. deux

  exemple a) un de deux un de deux

  exemple b) deux de deux

A-1 un

A-2 deux

```
\begin{enumerate}[EX i.]
\item un un un un un un un
      un un un un un un un\label{LA}
\item deux
    \begin{enumerate}[exemple a)]
    \item un de deux  un de deux\label{LB}
    \item deux de deux
    \end{enumerate}
\end{enumerate}

\begin{enumerate}[{A}-1]
\item un\label{LC}
\item deux
\end{enumerate}
```

Les commandes `\label` et `\ref` peuvent être utilisées comme dans l'environnement enumerate standard. `\ref` produit seulement la valeur du compteur, et non l'étiquette complète. L'affichage est fait dans le même style que `\item`, qui a été déterminé par la présence de l'un des caractères `A a I i 1` dans l'argument optionnel. Dans l'exemple ci-dessus `\ref{LA}`, `\ref{LB}` et `\ref{LC}` produisent « i », « iia » et « 1 » respectivement.

*Ce fichier a le numéro de version v3.00, révisé le 1999/03/05.

†Dernière mise à jour le 20/01/2000

2 Macros

```
⟨*package⟩
```

`\@enlab` Internal token register used to build up the label command from the optional argument.

```
\newtoks\@enLab
```

`\@enQmark` This just expands to a '?'. `\ref` will produce this, if no counter is printed.

```
\def\@enQmark{?}
```

The next four macros build up the command that will print the item label. They each gobble one token or group from the optional argument, and add corresponding tokens to the register `\@enLab`. They each end with a call to `\@enloop`, which starts the processing of the next token.

`\@enLabel` Add the counter to the label. `#2` will be one of the 'special' tokens `A a I i 1`, and is thrown away. `#1` will be a command like `\Roman`.

```
\def\@enLabel#1#2{%
  \edef\@enThe{\noexpand#1{\@enumctr}}%
  \@enLab\expandafter{\the\@enLab\csname the\@enumctr\endcsname}%
  \@enloop}
```

`\@enSpace`
`\@enSp@ce` Add a space to the label. The tricky bit is to gobble the space token, as you can not do this with a macro argument.

```
\def\@enSpace{\afterassignment\@enSp@ce\let\@tempa= }
\def\@enSp@ce{\@enLab\expandafter{\the\@enLab\space}\@enloop}
```

`\@enGroup` Add a `{ }` group to the label.

```
\def\@enGroup#1{\@enLab\expandafter{\the\@enLab{#1}}\@enloop}
```

`\@enOther` Add anything else to the label

```
\def\@enOther#1{\@enLab\expandafter{\the\@enLab#1}\@enloop}
```

`\@enloop`
`\@enloop@` The body of the main loop. Eating tokens this way instead of using `\@tfor` lets you see spaces and **all** braces. `\@tfor` would treat `a` and `{a}` as special, but not `{{a}}`.

```
\def\@enloop{\futurelet\@entemp\@enloop@}
```

```
\def\@enloop@{%
  \ifx A\@entemp         \def\@tempa{\@enLabel\Alph  }\else
  \ifx a\@entemp         \def\@tempa{\@enLabel\alph  }\else
  \ifx i\@entemp         \def\@tempa{\@enLabel\roman }\else
  \ifx I\@entemp         \def\@tempa{\@enLabel\Roman }\else
  \ifx 1\@entemp         \def\@tempa{\@enLabel\arabic}\else
  \ifx \@sptoken\@entemp \let\@tempa\@enSpace        \else
  \ifx \bgroup\@entemp   \let\@tempa\@enGroup        \else
  \ifx \@enum@\@entemp   \let\@tempa\@gobble         \else
                         \let\@tempa\@enOther
```

Hook for possible extensions

```
                         \@enhook
```

```
             \fi\fi\fi\fi\fi\fi\fi\fi
```

Process the current token, then look at the next.

```
  \@tempa}
```

`\@enhook` Hook for possible extensions. Some packages may want to extend the number of special characters that are associated with counter representations. This feature was requested to enable Russian alphabetic counting, but here I give an example of a footnote symbol counter, triggered by `*`.

To enable a new counter type based on a letter, you just need to add a new `\ifx` clause by analogy with the code above. So for example to make `*` trigger footnote symbol counting. a package should do the following.

Initialise the hook, in case the package is loaded before enumerate.

```
\providecommand\@enhook{}
```

Add to the hook a new `\ifx` clause that associates `*` with the `\fnsymbol` counter command.

```
\g@addto@macro\@enhook{%
  \ifx *\@entemp
    \def\@tempa{\@enLabel\fnsymbol}%
  \fi}
```

This code sequence should work whether it is loaded before or after this enumerate package. Any number of new counter types may be added in this way.

At this point we just need initialise the hook, taking care not to over write any definitions another package may already have added.

```
\providecommand\@enhook{}
```

`\enumerate` The new `enumerate` environment. This is the first half of the original enumerate environment. If there is an optional argument, call `\@@enum@` to define the label commands, otherwise call `\@enum@` which is the second half of the original definition.

```
\def\enumerate{%
  \ifnum \@enumdepth >3 \@toodeep\else
      \advance\@enumdepth \@ne
      \edef\@enumctr{enum\romannumeral\the\@enumdepth}\fi
  \@ifnextchar[{\@@enum@}{\@enum@}}
```

`\@@enum@` Handle the optional argument..

```
\def\@@enum@[#1]{%
```

Initialise the loop which will break apart the optional argument. The command to print the label is built up in `\@enlab`. `\@enThe` will be used to define `\theenum` n.

```
  \@enLab{}\let\@enThe\@enQmark
```

The `\@enum@` below is never expanded, it is used to detect the end of the token list.

```
  \@enloop#1\@enum@
```

Issue a warning if we did not find one of the ‘special’ tokens.

```
  \ifx\@enThe\@enQmark\@warning{The counter will not be printed.%
   ^^J\space\@spaces\@spaces\@spaces The label is: \the\@enLab}\fi
```

Define `\labelenum` n and `\theenum` n.

```
  \expandafter\edef\csname label\@enumctr\endcsname{\the\@enLab}%
  \expandafter\let\csname the\@enumctr\endcsname\@enThe
```

Set the counter to 7 so that we get the width of 'vii' if roman numbering is in force then set `\leftmargin` n. to the width of the label plus `\labelsep`.

```
\csname c@\@enumctr\endcsname7
\expandafter\settowidth
          \csname leftmargin\romannumeral\@enumdepth\endcsname
          {\the\@enLab\hspace{\labelsep}}%
```

Finally call `\@enum@` which is the second half of the original definition.

```
\@enum@}
```

`\@enum@` All the list parameters have now been defined, so call `\list`. This is taken straight from the original definition of `\enumerate`.

```
\def\@enum@{\list{\csname label\@enumctr\endcsname}%
          {\usecounter{\@enumctr}\def\makelabel##1{\hss\llap{##1}}}}
```

```
⟨/package⟩
```