The enumerate package[*]

David Carlisle
carlisle@cs.man.ac.uk

1994/01/31

Abstract

This package gives the enumerate environment an optional argument which determines the style in which the counter is printed.

An occurence of one of the tokens `A` `a` `I` `i` or `1` produces the value of the counter printed with (respectively) `\Alph` `\alph` `\Roman` `\roman` or `\arabic`.

These letters may be surrounded by any strings involving any other TEX expressions, however the tokens `A` `a` `I` `i` `1` must be inside a `{ }` group if they are not to be taken as special.

1 Examples

EX i. one one one one one one one one one one one one

EX ii. two

  example a) one of two one of two one of two

  example b) two of two

A-1 one

A-2 two

```
\begin{enumerate}[EX i.]
\item one one one one one one one
       one one one one\label{LA}
\item two
    \begin{enumerate}[{example} a)]
    \item one of two  one of two
         one of two\label{LB}
    \item two of two
    \end{enumerate}
    \item two of two
    \end{enumerate}
\end{enumerate}

\begin{enumerate}[{A}-1]
\item one\label{LC}
\item two
\end{enumerate}
```

`\label` and `\ref` may be used as with the standard `enumerate` environment. `\ref` only produces the counter value, not the whole label. `\ref` prints the value in the same style as `\item`, as determined by the presence of one of the tokens `A a I i 1` in the optional argument. In the above example `\ref{LA}`, `\ref{LB}` and `\ref{LC}` produce 'i', 'iia' and '1' respectively.

[*]This file has version number v2.02, last revised 1994/01/31.

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

Process the current token, then look at the next.

```
  \@tempa}
```

\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⟩
```