The fancynum package*

J. J. Green

2000/08/08

1 Introduction

The fancynum package is designed to aid the typesetting of numbers, particularly (but not exclusively) the ASCII representation of floating point numbers as written by computers.

In the sequel we refer to the glyph used to separate the integer and decimal parts of a decimal number as the *decimal symbol*, and that used to group the digits of the integer and decimal parts as the *group symbol*.

The author invites suggestions on improvements to the package. In particular, any information on the typographic conventions for setting numbers in different languages would be most welcome.

2 Usage

The fancynum package is quite standard in its usage. After including the package in the preamble, the macro `\fnum` is available in mathematics mode. A call to `$\fnum{3.141593e+05}$` will be set as $3{\cdot}141{,}593 \times 10^5$, and so on. Further examples can be found in the file `examples.tex` included in the distribution.

The operation of the `\fnum` macro can be modified with the commands `\setfnumdsym`, `\setfnumgsym` and `\setfnummsym`, which set the value of the decimal, group and multiplication symbols, respectively.

3 Limitations

The `\fnum` has some limitations on the form of its argument:

- A decimal must have at least one digit either side of the decimal symbol;
- The optional exponential symbol must be `e`;
- If the exponent must have at least one digit.

Consequently the strings `.312e20`, `312.e20` and `3.142E20` are not acceptable arguments. I hope to remove these restrictions at some point.

*This file has version number 0.92, last revised 2000/08/08.

4 Package Options

The package supports a small number of options. The locale-specific options set default values for all of the typographic parameters. The **english** option uses the centred dot as the decimal symbol and a comma as the group symbol. The **french** option uses the comma as the decimal symbol and a lower dot as the group symbol. Compare the English 3·141,593 against the French 3,141.593. If no local-specific option is specified then English will be taken as the default.

The remaining options give specific values over the typographic parameters, and override the locale-specific options.

The **tight** and **loose** options specify space around the $\times$ in the setting of numbers in exponential form. Compare the loose 2.3×10^3 with the tight 2.3×10^3. This option is useful where space is at a premium, or may be preferred for æsthetic reasons.

The **commas**, **thinspaces** and **plain** options specify the group symbol to be used, respectively `{,}`, `\,` and `\relax`.

5 Thanks

I am grateful to Heiko Oberdiek, Norman Gray, Michael Downes and Donald Arseneau for their assistance on `comp.text.tex`, and to Jerzy Kucharczyk for providing a bug report.

6 Implementation

```
⟨*package⟩
```

6.1 Typographic parameters

The decimal, group and multiplication symbols are stored in the global variables `\fn@decimalsym` etc. The following macros give access to the values used in the package.

```
\def\setfnumdsym#1{\gdef\fn@decimalsym{#1}}
\def\setfnumgsym#1{\gdef\fn@groupsym{#1}}
\def\setfnummsym#1{\gdef\fn@multsym{#1}}
```

6.2 Package options

The package options are simply calls the macro for the appropriate parameter.

```
\DeclareOption{english}{\setfnumdsym{{\cdot}}\setfnumgsym{{,}}}
\DeclareOption{french}{\setfnumdsym{{,}}\setfnumgsym{{.}}}
\DeclareOption{tight}{\setfnummsym{{\times}}}
\DeclareOption{loose}{\setfnummsym{\times}}
\DeclareOption{commas}{\setfnumgsym{{,}}}
\DeclareOption{thinspaces}{\setfnumgsym{\,}}
\DeclareOption{plain}{\setfnumgsym{\relax}}
```

We set the default values before processing the options

```
\ExecuteOptions{english,loose}
\ProcessOptions\relax
```

6.3 Utility macros

`\fn@length` Find the length of a string mod 3 (taken directly from the example on p. 219 of the TEXbook). Here though, a call to `\fn@length` assigns the calculated value the (global) variable `\fn@strlen`.

```
\newcount\fn@strlen
\def\fn@length#1{\fn@strlen=0 \fn@getlength#1\end}
\def\fn@getlength#1{\ifx #1\end \let\next=\relax \else
  \advance\fn@strlen by1
  \ifnum\fn@strlen=3 \fn@strlen=0 \fi
  \let\next=\fn@getlength\fi \next}
```

6.4 Typesetting

6.4.1 The decimal

`\fn@fracpunct` Punctuate the fractional part of a decimal. An easy cyclic recursion.

```
\def\fn@fracpunct#1{\fn@fpa#1@ }
\def\fn@fpa#1#2#3 {#1\if #2@ \else \fn@fpb#2#3 \fi}
\def\fn@fpb#1#2#3 {#1\if #2@ \else \fn@fpc#2#3 \fi}
\def\fn@fpc#1#2#3 {#1\if #2@ \else \fn@groupsym\fn@fpa#2#3 \fi}
```

`\fn@intpunct` Punctuate the integer part of a decimal. This is not as easy as the fractional part since we need to know the length before we start (no doubt there is a direct recursive method to do this, but I'm not clever enough to work it out).

```
\def\fn@intpunct#1{\fn@ipa#1 }
```

The macro `\fn@ipa` prints the string's initial ± and passes on the rest of the string to the `\fn@ipb` macro. It might be worthwhile stripping the redundant '+' here, but I am minded not to do this since whether it is present is the choice of the user (unlike the redundancy in the sign of the exponent — see below).

```
\def\fn@ipa#1#2 {%
  \if +#1 +\fn@ipb#2 \else
  \if -#1 -\fn@ipb#2 \else
  \fn@ipb#1#2 \fi \fi}
```

The `\fn@ipab` macro finds the length (modulo 3) with `\fn@strlen` and calls the appropriate output macro.

```
\def\fn@ipb#1 {\fn@length{#1}
  \ifcase\fn@strlen
  \fn@ipc#1 \or
  \fn@ipd#1 \or
  \fn@ipe#1 \fi}
```

The `\fn@ip[cde]` macro prints the first couple of characters and then calls `\fn@ipf` which calls `\fn@fpa` (see above) to finish the job.

```
\def\fn@ipc#1#2#3#4 {#1#2#3\fn@ipf#4@ }
\def\fn@ipd#1#2     {#1\fn@ipf#2@ }
\def\fn@ipe#1#2#3   {#1#2\fn@ipf#3@ }
\def\fn@ipf#1#2     {\if #1@ \else \fn@groupsym\fn@fpa#1#2 \fi}
```

`\fn@decimal` Set a proper decimal. The pattern matching trick used here (and for the exponent) is due to a conversation between Heiko Oberdiek and Michael Downs on `comp.text.tex`. See the file `ctt.txt` included in the distribution for details.

```
\def\fn@propdecimal#1#2{\fn@intpunct{#1}\fn@decimalsym\fn@fracpunct{#2}}
\def\fn@impropdecimal#1#2{\fn@intpunct{#1}}
\def\fn@decimalsplit#1.#2.#3#4#5{#4{#1}{#2}}
\def\fn@decimal#1{\fn@decimalsplit#1..\fn@propdecimal\fn@impropdecimal\empty}
```

6.4.2 The exponent

`\fn@signedint` Set a signed exponent. The C standard I/O library functions *printf*, *fprintf*, etc., as well as many Fortran compilers, write a '+' in the exponent if it is positive. Since this is superfluous, both typographically and mathematically, we remove it if we find it.

```
\def\fn@signedint#1#2 {%
  \if +#1
    \fn@unsignedint#2\relax
  \else
    \if -#1
      -\fn@unsignedint#2\relax
    \else
       \fn@unsignedint#1#2\relax
    \fi
  \fi}
```

`\fn@unsignedint` Set an unsigned exponent. Here we remove leading zeros.

```
\def\fn@unsignedint#1#2{%
  \ifx #2\relax \let\next=#1
  \else
    \ifx 0#1 \let\next=\fn@unsignedint
    \else
    \let\next=#1
    \fi
  \fi
  \next #2}
```

6.4.3 The general number

`\fnum` The setting of an floating point number. Our processing is dependent on whether the argument contains the letter 'e'.

```
\def\fn@exp#1#2{\fn@decimal{#1}\fn@multsym10^{\fn@signedint#2 }}
\def\fn@noexp#1#2{\fn@decimal{#1}}
\def\fn@realsplit#1e#2e#3#4#5{#4{#1}{#2}}
\def\fnum#1{\fn@realsplit#1ee\fn@exp\fn@noexp\empty}
```

⟨/package⟩