Documentation for multido.tex:
A loop macro for Generic TeX

Timothy Van Zandt*

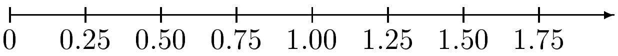
Version 1.3
6 November 1992

1 Usage notes

`multido.tex/multido.sty` contains the `\multido` macro, which is a loop facility for Generic TeX. This macro happens to be useful for drawing pictures, and was originally developed for the PSTricks package,[1] but you can use it for other purposes as well.

A special feature is support of fixed-point addition. For example, PSTricks uses the `\multido` to put numbers on axes, much like in the following LaTeX example: PSTricks uses `\multdo` internally to put numbers on axes, much like in this LaTeX example:

```
\setlength{\unitlength}{1cm}
\small
\begin{picture}(8,1)(0,-.5)
\put(0,0){\vector(1,0){8}}
\multido{\i=0+1,\n=0+0.25}{8}{%
  \put(\i,-.1){\line(0,1){.2}}
  \put(\i,-.2){\makebox[t](0,0){\n}}}
\end{picture}
```



The general syntax for `\multido` is:

`\multido`{*variables*}{*repetitions*}{*stuff*}

*Author's address: Department of Economics, Princeton University, Princeton, NJ 08544-1021, USA. Internet: `tvz@Princeton.EDU`

[1]PSTricks is an extensive collection of PostScript-based macros for Generic TeX. It is available from the /pub directory at Princeton.EDU, and TeX archives.

stuff is whatever you want repeated; it can be any balanced TEX input. *repetitions* is the number times *stuff* is repeated.

The first argument is the interesting one. *variables* is a comma-separated list of variable declarations.[2] Each variable declaration is of the form:

variable `=` *initial value* `+` *increment*

variable is a command sequence that can be used in *stuff*. It is initially set to *initial value*, and is then incremented by *increment* with each repetition.

The first letter of the variable name determines the variable type. There are four variable types:

Dimension (d or D) The initial value and the increment should be dimensions (lengths, in LATEX parlance). The substitution text is a dimension, with `sp` units. E.g., `\dx=4cm+5pt`.

Number (n or N) The initial value and increment should be integers or numbers with the same number of digits to the right of the decimal. The one exception is that it is always OK for the initial value to be an integer. There can be at most 8 digits on each side of the decimal. The substitution text is a number, with fixed-point addition. E.g., `\n=3+7.05`, `\Nx=5.30+-1.25`.

Integer (i or I) The initial value and increment should be integers. This gives the same result as using a number variable, but it is faster. E.g., `\I=2+-1`.

Real (r or R) The initial value and increment should be integers or numbers with at most 4 digits on each side of the decimal. The substitution text is a number, but with floating point addition and occasional small errors. This gives a less satisfactory result than using a number variable, but it is faster. E.g., `\ry=4.2+1.05`.

Here are some examples that illustrate how the substitution text is determined:

`\multido{}{10}{\TeX\ }`

TEX TEX TEX TEX TEX TEX TEX TEX TEX TEX

`\multido{\d=2pt+3pt}{5}{\d, }`

131072sp, 327680sp, 524288sp, 720896sp, 917504sp,

`\multido{\n=2+3}{10}{\n, }`

2, 5, 8, 11, 14, 17, 20, 23, 26, 29,

`\multido{\i=2+3}{10}{\i, }`

2, -1, -4, -7, -10, -13, -16, -19, -22, -25,

[2] Don't use commas to mark the decimal point within the *variables* argument, as they will be confused for delimiters.

```
\multido{\r=2+3.05}{6}{\r, }
```

2.0, 5.05, 8.1, 11.15001, 14.20001, 17.25002,

```
\multido{\n=2.00+3.05}{8}{\n, }
```

2.00, -1.05, -4.10, -7.15, -10.20, -13.25, -16.30, -19.35,

Here are some details about the choice of names:

- Your computer won't explode if you use names that conflict with TEX internal commands, but you might want to check name conflicts if you get inexplicable errors. The command `\MultidoCheckNames` can be useful in this case. It causes `\multido` to report an error whenever you use a variable name that is already defined. But see the next item.
- The whole `\multido` loop is grouped. This means, e.g., that although `\i` is a Plain TEX command sequence (giving a dotless "ı"), you can use the variable `\i` if you do not use any dotless i's in *stuff* (and if you do not use `\MultidoCheckNames`).

Here are a few more details:

- `\Multido` commands can be nested.
- Spaces after a `\multido` command are ignored. This makes `\multido` more hospitable for pictures.
- Spaces between the various parts of the *variables* argument are ignored.

And finally here a few special features, some of which are of interest mainly macro writers and other TEXnicians:

- The material that is repeated is not grouped, so that you can insert your own recursive routines.
- There is a variant, `\mmultido`, which works just like `\multido` except that the variables are all incremented once before starting.
- There are variants, `\Multido` and `\MMultido` of `\multido` and `\mmultido`, resp., that do not group the whole loop. This can be useful, e.g., for making entries in an alignment environment. However, these cannot be nested within any `\multido` macro.
- If the number of repetitions is a negative number, the variables are incremented backwards.
- The count register `\multidocount` keeps track of the number of the iterations.

- The command `\multidostop` causes the `\multido` loop to quit at the end of the current iteration.

- Fixed point addition is performed by `\FPadd` and `\FPsub`:

  `\FPadd`{*num1*}{*num2*}{*cs*}
  `\FPsub`{*num1*}{*num2*}{*cs*}

  num2 is added to or subtracted from *num1*, and the answers is stored in the command sequence given as the third argument. The rules about decimals and so on that apply to number variables apply here as well. E.g., after

  ```
  \FPsub{1.75}{-0.15}{\answer}
  ```

  the definition of `\answer` is 1.90.