NAME

mathsPIC

AUTHORS

A. Syropoulos and R.W.D. Nickalls (April 26, 2010)

asyropoulos[at]<yahoo><com>
dick[at]<nickalls><org>

DESCRIPTION

mathsPIC is a Perl filter program for PiCTeX. mathsPIC has its own macro and macro library capability, and allows use of mathsPIC, PiCTeX, TeX and LaTeX commands. A significant feature of mathsPIC is that it allows access to the command-line, and so allows the user to extend mathsPIC commands by calling Perl and other programs written to perform particular drawing actions. See the package manual for full details and examples. The latest version can be downloaded from

CTAN: tex-archive/graphics/pictex/mathspic/perl

Commands which can be used in the mathsPIC script file fall into four main groups (a) mathsPIC macro commands (prefixed with %def), (b) regular mathsPIC commands (do not have a backslash), (c) regular PiCTeX commands (all have a backslash), and (d) regular TeX and LaTeX commands (all have a backslash).

The following mathematics functions can used (note that decimal fractions having an absolute value less than 1 must have a leading zero). Note also that all the trignometric functions require their argument in radians.

Trigonometric: sin(), cos(), tan(), asin(), acos(), atan()

Remainder: rem(); eg var r=12 rem(5)

Integer: int(); eg var r= int(3.87) --> 3

Sign (returns -1, 0, +1): sgn(); eg var s=sgn(-3.27) --> -1

Square root: sqrt(); eg var s = sqrt(14)

Exponentiation: **; eg var j = r**2

Pi constant (3.14159...): _Pi_ and _pi_

e constant (2.71828...): _E_ and _e_

Linethickness: _linethickness_ ; eg var t = _linethickness_

COMMAND-LINE USE

perl mathspic.pl [-b] [-c] [-h] [-o <outfile>] <infile>

-b enables beep if mathsPIC detects an error

-c disables the writing of comments to output file

-h displays the help file

-o defines the output file name

MACRO COMMANDS

macro definition commands are prefixed with %def and can take either 0, 1, or more parameters. Macros will generally be used as part of a **var** command as shown below. Macros are deleted using the **%undef** command.

```
-----syntax:
%def MACRONAME(parameters)<macrodefinition>
%undef MACRONAME(parameters)
```

-----notes:
Notes: (a) the () must be used in the definition even if no parameters are used, (b) the name can be any combination of upper and lower case characters and numbers, (c) when the macro is used in a command it is prefixed by a & symbol, (d) it is a good idea to always place a % symbol at the end of the definition, (e) comments (prefixed by a % symbol) can be placed after the macro definition just as in TeX or LaTeX.

```
-----examples:
%def d2r()_pi_/180%            % degrees2radians
%def AreaOfRectangle(x,y)x*y%    % width x, length y
%undef d2r()                   % delete the macro

-----use:
var j2= 6*(&d2r(45) + 23)
var a3 = 3*&AreaOfRectangle(5,7)
```

GENERAL COMMANDS

NUMERICAL EXPRESSIONS

When dealing with commands we will refer frequently to the term ‘numerical expression’ by which is meant either (a) a number (integer or decimal), (b) a numeric variable or constant (defined using the var or const command), (c) any mathsPIC function, macro, or mathematical expression which evaluates to a number, or (d) a pair of point names (e.g. AB) representing the Pythagorean distance between the two points. A leading zero must be used with decimal fractions less than one.

In general, if a command’s argument accepts a number then it will also accept a ‘numerical expression’ (<expr>) as defined above. Sometimes a following <unit> is associated with the number or numerical expression, in which case the number or numerical expression can be delimited by a round bracket (or separated from the unit by a <space>), as shown in the following examples.

```
-----examples:
ArrowShape(3mm, 20,40)
var h=4
```

```
ArrowShape(h mm, 20, 40)
ArrowShape((2*h)mm,20,40)
```

BACKSLASH \

A leading backslash without a following space indicates that it is part of a PiCTeX, TeX or LaTeX command, in which case mathsPIC simply copies the whole line verbatim into the output file. A leading backslash followed by one or more spaces makes mathsPIC copy the whole line verbatim into the output file but without the backslash.

USING THE COLOR PACKAGE

The standard COLOR package can be used with mathsPIC, but note that it is important to load the COLOR package after the mathsPIC package.

It is best to place a comment symbol % at the end of LaTeX and TeX commands to limit white space at the end.

In the event of any colour-spill from a diagram into any following text (this used to be a problem in early TeX implementations) consider using the \normalcolor command as a delimiter within the \beginpicture...\endpicture environment.

==============================

ARROWSHAPE

This command defines the shape of an arrowhead, and allows different arrowheads to be customised.

The default arrow shape is equivalent to the Arrowshape(2mm,30,40) command. This default arrowhead shape can be reset using the Arrowshape(default) command, as shown in the following example.

```
-----syntax:
arrowshape(<length>[units], <angledeg>, <angledeg>)

-----examples:
Arrowshape(4mm,30,60)
drawArrow(AB)
Arrowshape(default)
```

==============================

beginLOOP...endLOOP

This is an environment which cycles a block of code a specified number of times.

```
-----syntax:
beginLoop <expr>
...
endLoop
```

-----notes:
The block of code which lies within the environment is input <expr> times.

-----example:
```
beginLoop 5
...
endLoop
```

==============================

beginSKIP...endSKIP
This is an ‘environment’ within which commands are not actioned. It is useful in development for testing isolated commands and excluding other commands.

==============================

CONST
The const command is used to define scalar constants. Note that a constant-name must begin with a single letter (either upper or lower case), and may have up to a maximum of three following digits. Note that constants, variables and points have the same name structure, and a constant could have the same name as a point (and so we suggest points have uppercase letters and variables and constants have lowercase letters). The scalar argument can be any numeric expression. New values cannot be re-allocated to existing constant-names. If this occurs mathsPIC will issue an error message.

-----syntax:
```
const name = <expr>
```

-----examples:
```
const r = 20, r4 = r3*tan(0.3)
```

==============================

DashArray
The dasharray command takes an arbitrary number of paired arguments that are used to specify a dash pattern.

-----syntax
```
dasharray(d1 , g1 , d2 , g2 , ... )
```

-----notes
The ds denotes the length of a dash and the gs denotes the length of the gap between two consecutive dashes. There must be an even number of arguments. If a variable or expression is used then it should be separated from the unit either by a <space> or with round brackets ( ) as shown below.

-----example

```
dasharray(6pt, 2pt, 1pt, 2pt)
var d=2
dasharray(6pt, 2pt, 1pt, d pt)
dasharray(6pt, 2pt, 1pt, (d)pt)
dasharray(6pt, 2pt, 1pt, (3*d)pt)
```

==============================

DrawAngleArc
This command draws an arc in the specified angle, a distance <radius> from the angle. The angle is either <internal> (less than 180 deg) or <external> (greater than 180 deg). The direction of the arc is either <clockwise> or <anticlockwise>, and this direction must correspond with the letter sequence specified for the angle. Strange and unexpected results will be produced if the four parameters are not internally consistent. The option order angle/radius/internal or external/clockwise or anticlockwise is important. The <radius> parameter can be any numerical expression.

```
-----syntax:
DrawAngleArc{angle(), radius(), external, clockwise}

-----example:
DrawAngleArc{angle(ABC), radius(3), external, clockwise}
var r=3
DrawAngleArc{angle(ABC), radius(r), external, clockwise}
```

==============================

DrawAngleArrow
This command draws a curved arrow in the specified angle, a distance <radius> from the angle. The angle is either <internal> (less than 180 deg) or <external> (greater than 180 deg). The direction of the arrow is either <clockwise> or <anticlockwise>, and this direction must correspond with the letter sequence specified for the angle. Strange and unexpected results will be produced if the four parameters are not internally consistent. The option order angle/radius/internal/clockwise is important. The <radius> parameter can be any numerical expression.

```
-----syntax:
DrawAngleArrow{angle(), radius(), external, clockwise}

-----example:
DrawAngleArrow{angle(ABC), radius(3), external, clockwise}
var r=3
DrawAngleArrow{angle(ABC), radius(r), external, clockwise}
```

==============================

DrawArrow
This command draws an arrow(s) joining two points. The direction of the arrow is in the point order specified.

-----syntax:
drawArrow(<line> [,<line>] ...)

-----notes:
The length option can only refer to one arrow

-----example:
drawArrow(AB)
drawArrow(FG, HJ)

==============================

DrawCircle
This command draws a circle defined by its radius and the point-name of its centre. The <radius> can be any numerical expression. If the units of the X and Y axes are different, circles may be drawn strangely, and mathsPIC therefore generates a warning message to this effect.

-----syntax:
DrawCircle(<center>, <radius>)

-----examples:
drawCircle(C2,5)
drawCircle(C2,r2)
drawCircle(C2,r2/tan(1.3))
drawCircle(C2,AB)

==============================

DrawCircumcircle
This command draws the circumcircle of a triangle.

-----syntax:
DrawCircumcircle(<triangle>)

-----example:
drawCircumcircle(ABC)

==============================

DrawCurve
This command draws a smooth quadratic curve through three points in the point order specified. Note that curves drawn using this command do not break to avoid line-free zones associated with the points.

-----syntax:
DrawCurve(<point><point><point>)

-----example:

drawCurve(ABC)

==============================

DrawExcircle
This command draws the excircle touching one side of a triangle.

-----syntax:
DrawExcircle(<triangle>, <side>)

-----example:
drawExcircle(ABC, BC)

==============================

DrawIncircle
This command draws the incircle of a triangle.

-----syntax:
DrawIncircle(<triangle>)

-----example:
drawIncircle(ABC)

==============================

DrawLine
This command draws a line joining two or more points. Use the Linethickness command to vary thickness. This command uses the PiCTeX \putrule command for horizontal and vertical lines, and the \plot command for all other orientations.

-----syntax:
DrawLine(<points> [, <points>])

-----notes:
<points> is any sequence of two or more point names.
<expr> is any numerical expression.
Lines are drawn in the order specified.
Lines are separated by a comma.

-----examples:
drawline(AB)
drawline(BCDE)
drawline(FG, HJK, PQRST)

==============================

DrawPerpendicular

This command draws the perpendicular from a point to a line.

```
-----syntax:
DrawPerpendicular(<point>, <line>)

-----example:
drawPerpendicular(P,AB)
```

==============================

DrawPoint

This command draws the point-symbol at the point-location. Commas must not be used to separate point names. The default point-symbol is bullet unless an optional point-symbol (or string of characters) is specified in the associated point command.

```
-----syntax:
DrawPoint(<point> [<point> ..])

-----examples:
drawpoint(T4)
drawpoint(ABCDEF)
drawpoint(P1 P2 P3 P4)
```

==============================

DrawRightangle

This command draws the standard right-angle symbol in the internal angle specified at the size specified by <expr>.

```
-----syntax:
DrawRightangle(<angle>, <expr>)
```

-----notes:
The <expr> can be any numerical expression.

```
-----example:
drawRightangle(ABC,3)
drawRightangle(ABC,PQ)
var d=5
drawRightangle(ABC,d)
```

==============================

DrawSquare

This command draws a square defined by its side and the point-name of its centre. The <side-length> can be any numerical expression.

-----syntax:

```
 DrawSquare(<centerpoint>, <sidelength>)

 -----examples:
 drawSquare(P,5)
 var s2=3, j=2
 drawSquare(P,s2)
 drawSquare(P, s2*4/(3*j))
 drawSquare(P,AB)
```

==============================

DrawThickArrow
This command draws a thick arrow(s) joining two points. The direction of the arrow is in the point order specified. The shape of the arrowhead is controlled by the ArrowShape command.

```
 -----syntax:
 drawThickArrow(<line> [,<line>,...])

 -----examples:
 drawThickarrow(BC)
 drawThickarrow(PQ, RS)
```

==============================

DrawThickLine
This command draws a thick line(s) joining two points. The direction of the line is in the point order specified. Use the Linethickness command to vary thickness of a line.

```
 -----syntax:
 drawThickLine(<line> [,<line>,...])

 -----examples:
 drawThickline(BC)
 drawThickline(PQ, RS)
```

==============================

InputFile
This command inputs a plain text file containing mathsPIC commands. Optionally, the file can be input several times, in which case this command functions like a DO--LOOP. The <loopnumber> can be any numerical expression. If the <loopnumber> is not an integer then mathsPIC will round the value down to the nearest integer. See also the beginLOOP ... endLOOP commands.

```
 -----syntax:
 inputFile[*](<filename>)[<loopnumber>]
```

-----notes:
The inputfile* command is used to input a file in verbatim, i.e. a file with no mathsPIC

commands, for example, a file containing only PiCTeX commands or data-points for plotting etc. Note that the inputfile* command has no <loopnumber> option. Note also that PiCTeX requires a ODD number of points.

```
 -----examples:
 inputFile(myfile.dat)[4]
 inputFile*(mycurvedata.dat)
```

==============================

LineThickness

This command sets a particular linethickness. The command linethickness(default) restores the working linethickness to the default value of 0.4pt. The current value of the linethickness (in current units) can be accessed using the var command (this can be useful when drawing figures using thick lines) .

```
 -----syntax:
 LineThickness(<expr><units>)
 LineThickness(default)
 var t = _linethickness_
```

-----notes:

This command also sets the font to cmr and plotsymbol to \CM . and also sets the rule thickness for drawing horizontal and vertical lines. It is important to include a leading zero with decimal fractions less than one.

```
 -----examples:
 linethickness(2pt)
 var t=3
 linethickness((t)pt)
 lineThickness((2*t)pt)
 linethickness(default)
 var t = _linethickness_
```

-----caution:

Note that there is a similar PiCTeX command with the same name (but with a different syntax).

==============================

PAPER

Defines the plotting area in terms of the options units(), xrange(), yrange(), axes(), and ticks(). The units() argument must contain a numeric value and a valid TeX length unit mm, cm, pt, pc(pica), in(inch), bp(big point), dd(didot), cc(cicero), sp(scaled point). The X and Y axes can have different units (see second example below). The axes() arguments XYTBLR refer to the X and Y axes, and the Top, Bottom, Left and Right axes. A * following one of the axes disables ticks on that axis. The X and Y axes pass through the zeros.

-----examples:

```
paper{units(1cm),xrange(0,10),yrange(0,10)}
paper{units(2cm,1cm),xrange(0,10),yrange(0,10),axes(LB)}
paper{units(1mm),xrange(0,100),yrange(0,100),axes(XY)}
paper{units(1cm),xrange(-5,5),yrange(-5,5),axes(LRTBXY),ticks(1,1)}
paper{units(1cm),xrange(-5,5),yrange(-5,5),axes(LRT*B*)}
```

==============================

POINT

Defines a new point by allocating coordinates to a new point name. The * option re-allocates coordinates to an existing point name.

```
-----syntax:
POINT[*](<name>){<point>}[symbol=<chars>, radius=<expr>]
POINT[*](<name>){<location>}[symbol=<chars>, radius=<expr>]
```

-----notes:
<name> one leading letter plus maximum of three trailing digits
<chars> any TeX string allowed in an \hbox{}
<expr> any numerical expression
The polar(r,theta) option defaults to radians for the angle theta. To work in degrees then must append <deg> eg: polar(r,theta deg). Can use <direction()> and <directiondeg()> to replace theta. Note that the term vector(AB) means use same (r, theta) as AB.

```
-----examples:
point(A){5,5}
point(B2){22,46}[symbol=$\odot$]
point(B2){22,46}[symbol=circle(2),radius=5]
var r=3
point(B2){22,46}[symbol=square(3),radius=r]
point(B123){22,46}[radius=5]
point(D2){B2, shift(5,5)}
var s = 3
point(D2){B2, shift(2*s,4*s)}
point(D3){D2, polar(6,32 deg)}
point(D4){D2, polar(6,1.2 rad)}
point(D4){D2, polar(6, direction(AB))}      %% radians by default
point(D4){D2, polar(6, directiondeg(AB) deg)}
point(G2){Q, rotate(P, 23 deg)}
point(G2){Q, vector(AB)}
point(D2){intersection(AB,CD)}
point(F){PointOnLine(AB,5.3)}
point(G){perpendicular(P,AB)}
point(H){circumcircleCenter(ABC)}
point(J){incircleCenter(ABC)}
point(K){excircleCenter(ABC,BC)}
point*(A){6,3}
point*(P){Q}
```

```
 point*(B){B, shift(5,0)}
 point*(P){xcoord(J),ycoord(K)}
```

==============================

PointSymbol
 This command allows the default point-symbol \bullet (with zero line-free radius) to be changed. The PointSymbol command is particularly useful where a set of points uses the same point-symbol, for example, when drawing graphs. The point-symbol can be reset to the default \bullet using the command PointSymbol(default).

```
 -----syntax:
 PointSymbol(<symbol>, <line-free-radius>)
 PointSymbol(default)

 -----notes:
 The PointSymbol command only influences subsequent point commands.
 The optional square bracket of the point command overrides the PointSymbol command.

 -----examples:
 PointSymbol($\odot$, 0.7)
 PointSymbol(default)
```

==============================

SYSTEM
 This command allows the user to access the command line and execute standard Linux commands. A important use for this command is to run a Perl program.

```
 -----syntax:
 System("<command>")

 -----notes:
 The <command> string must be in inverted commas.

 -----example:
 system("dir > mydir-listing.txt")
 system("perl myperlprogram.pl")
```

==============================

SHOW....
 This command makes mathsPIC return the value of a calculation or specified parameter; for example, the value of a particular angle, or the length of a line. The result is shown in the outputfile as a commented line. This allows mathsPIC commands to be adjusted in the light of calculations. There are currently five such commands as follows.

```
 -----syntax:
```

```
showLength(AB)
showAngle(ABC)      % returns angle in radians
showAngledeg(ABC)   % returns angle in degrees
showArea(ABC)
showPoints
showVariables
```

=============================

TEXT

This command places a text-string at a specific location. By default the text is centered vertically and horizontally at the specified point. Optionally, text can be placed relative to a point using appropriate combinations of the PiCTeX 'position' options l t r B b to align the (l)eft edge, (r)ight edge, (t)op edge, (B)aseline, (b)ottom edge respectively of the text box with the point-location.

Remember that the default units for the angle argument of the polar() expression is radians; hence you MUST append 'deg' if you want to work in degrees

```
-----syntax:
text(<string>){<location>}[<position options>]
text(<string>){<pointname>, shift(<x>,<y>)}[]
text(<string>){<pointname>, polar(<r>,<angle>[rad])}[]

-----examples:
text(A){5,6}
text($A_1$){A1, shift(2, 2)}
text(Z2){Z2, shift(5, -5)}[tr]
text(Z3){Z2, polar(5, 20 deg)}[Br]
text(Z4){Z2, polar(5, 1.34 rad)}
text(\framebox{Z5}){Z5}
```

=============================

VAR

The var command is used to define scalar variables. It can be any numerical expression. A variable-name must begin with a single letter (either upper or lower case), and may have up to a maximum of four following digits. If a more detailed variable name is required, then a simple alternative is to use a mathsPIC macro---as any string can be allocated via macros (see the beginning of this chapter for details on macros).

Note that variables, constants and points have the same name structure, and a variable can have the same name as a point (and so we suggest points have uppercase letters and variables and constants have lowercase letters). New values can be re-allocated to existing variable-names; however, when this occurs then mathsPIC does not issue a warning message to hightlight this fact.

If it is important to be warned if a potential variable is accidentally reallocated then one should consider using the const command instead (since mathsPIC does generate an error message if a constant is reallocated).

```
-----syntax:
var  <name> = <expr>
```

-----notes:
In addition to the mathematical functions mathsPIC functions which can be used with the var command are:

```
angle(<three-points>)          % returns angle in radians
angledeg(<three-points>)       % returns angle in degrees
area(<three-points>)
xcoord(<point>)
ycoord(<point>)
direction(<two-points>)     % returns angular direction in radians
directiondeg(<two-points>)  % returns angular direction in degrees
```

```
-----examples:
var r = 20, r4 = r3*tan(0.3), j = (r*2e3)**2,  r5 = AB
var e = _e_, p1 = _Pi_
var t = _linethickness_  % returns linethickness in current units
var g137 = angle(ABC)    %(default: returns in radians)
var g = angledeg(ABC)    % angle in degrees
var h = area(ABC)
var x2 = xcoord(A), y2 = ycoord(A)
var m5 = 12 rem 3     % remainder after dividing by 3
var r1 = direction(PQ)   % in radians
var d1 = directiondeg(PQ)
```

==============================

SEE ALSO

The mathsPIC package manual and examples

BUGS

Please report bugs to Dick Nickalls (dick [AT] nickalls [dot] org) or to Apostolos Syropoulos