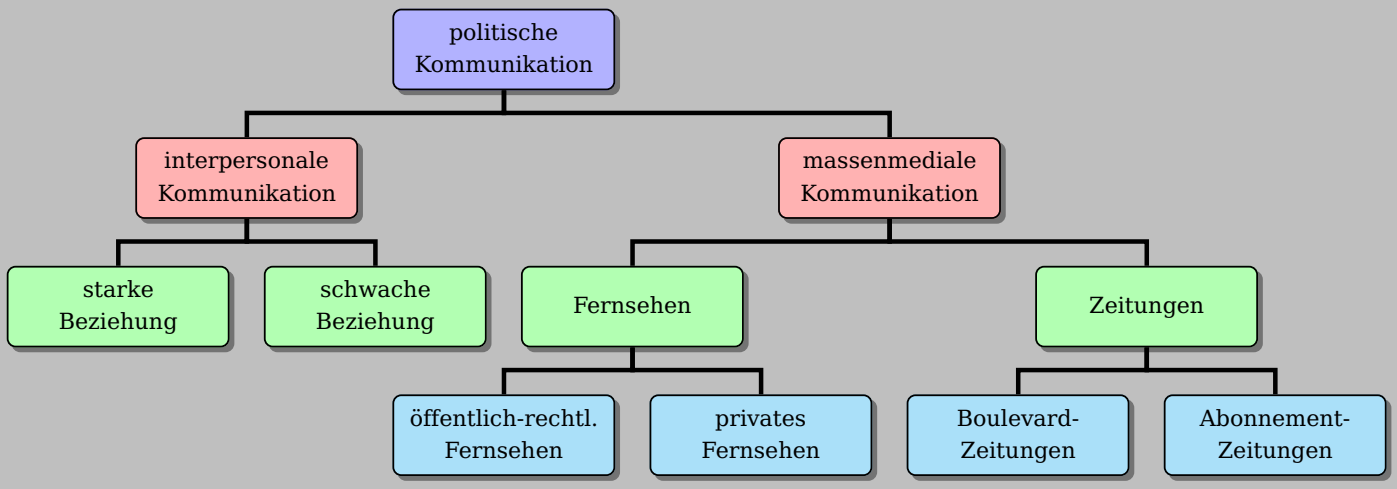
PSTricks

pst-node
Nodes and node connections

v.1.14

December 28, 2010



Package author(s):
Timothy Van Zandt
Michael Sharpe
Herbert Voß

This version of `pst-node` uses the extended keyval handling of pst-xkey and has a lot of the macros which were recently in the package `pstricks-add`. This documentation describes in the first part the basic node commands and connection from the old PSTricks documentation. The second part describes only the new and changed stuff. .

Thanks to: Marco Daniel; Denis Girou; Rolf Niepraschk; Sebastian Rahtz;

Contents

Part I.
Basic commands, connections and labels

The following pages are from the original documentation of `PSTricks` . This documentation file `pst-docfull` should be part of any TEX distribution and was inserted here with the following command:

```
\includepdf[pages=6-32]{pst-docfull}
```

Nodes and Node Connections

All the commands described in this part are contained in the file pst-node.tex/pst-node.sty.

The node and node connection macros let you connect information and place labels, without knowing the exact position of what you are connecting or where the lines should connect. These macros are useful for making graphs and trees, mathematical diagrams, linguistic syntax diagrams, and connecting ideas of any kind. They are the trickiest tricks in PSTricks!

There are three components to the node macros:

Node definitions The node definitions let you assign a name and shape to an object. See Section 6.

Node connections The node connections connect two nodes, identified by their names. See Section 7.

Node labels The node label commands let you affix labels to the node connections. See Section 8.

You can use these macros just about anywhere. The best way to position them depends on the application. For greatest flexibility, you can use the nodes in a **\pspicture**, positioning and rotating them with **\rput**. You can also use them in alignment environments. pst-node.tex contains a special alignment environment, **\psmatrix**, which is designed for positioning nodes in a grid, such as in mathematical diagrams and some graphs. **\psmatrix** is described in Section **??**. pst-node.tex also contains high-level macros for trees. These are described in Part **??**.

But don't restrict yourself to these more obvious uses. For example:

I made the file symbol a node. Now I can draw an arrow so that you know what I am talking about.

```
\rnode{A}{%
  \parbox{4cm}{\raggedright
    I made the file symbol a node. Now I can draw an
    arrow so that you know what I am talking about.}}
\ncarc[nodesep=8pt]{->}{A}{file}
```

6 Nodes

Nodes have a name. a boundary and a center.

The name is for refering to the node when making node connections and labels. You specify the name as an argument to the node commands. The name must contain only letters and numbers, and must begin with a letter. Bad node names can cause PostScript errors.

The center of a node is where node connections point to. The boundary is for determining where to connect a node connection. The various nodes differ in how they determine the center and boundary. They also differ in what kind of visable object they create.

Here are the nodes:

\rnode[*refpoint*]**{*name*}{*stuff*}**

\rnode puts *stuff* in a box. The center of the node is *refpoint*, which you can specify the same way as for **\rput**.

\Rnode*[*par*]**{*name*}{*stuff*}**

\Rnode also makes a box, but the center is set differently. If you align **\rnode**'s by their baseline, differences in the height and depth of the nodes can cause connecting lines to be not quite parallel, such as in the following example:

```
\Large
\rnode{A}{sp}  \hskip  2cm  \rnode{B}{Bit}
\ncline{A}{B}
```

sp———Bit

With **\Rnode**, the center is determined relative to the baseline:

```
\Large
\Rnode{A}{sp}  \hskip  2cm  \Rnode{B}{Bit}
\ncline{A}{B}
```

sp———Bit

You can usually get by without fiddling with the center of the node, but to modify it you set the

href=*num* **Default: 0**
vref=*dim* **Default: .7ex**

parameters. In the horizontal direction, the center is located fraction **href** from the center to the edge. E.g, if **href=-1**, the center is on the left edge of the box. In the vertical direction, the center is located distance **vref** from the baseline. The **vref**

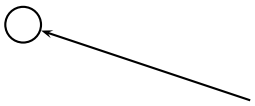
parameter is evaluated each time **\Rnode** is used, so that you can use ex units to have the distance adjust itself to the size of the current font (but without being sensitive to differences in the size of letters within the current font).

\pnode(*x*,*y*)**{*name*}**

This creates a zero dimensional node at (*x*,*y*).

\cnode*[*par*](*x*,*y*)**{*radius*}{*name*}**

This draws a circle. Here is an example with **\pnode** and **\cnode**:



```
\cnode(0,1){.25}{A}
\pnode(3,0){B}
\ncline{<-}{A}{B}
```

\Cnode*[*par*](*x*,*y*)**{*name*}**

This is like **\cnode**, but the radius is the value of

radius=*dim* **Default: 2pt**

This is convenient when you want many circle nodes of the same radius.

\circlenode*[*par*]**{*name*}{*stuff*}**

This is a variant of **\pscirclebox** that gives the node the shape of the circle.

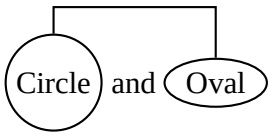
\cnodeput*[*par*]{*angle*}(*x*,*y*)**{*name*}{*stuff*}**

This is a variant of **\cput** that gives the node the shape of the circle. That is, it is like

```
\rput{angle}(x,y){\circlenode{name}{stuff}}
```

\ovalnode*[*par*]**{*name*}{*stuff*}**

This is a variant of **\psovalbox** that gives the node the shape of an ellipse. Here is an example with **\circlenode** and **\ovalnode**:



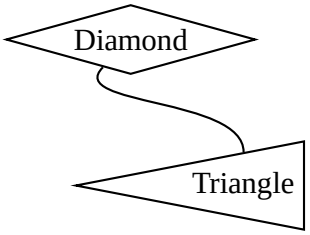

```
\circlenode{A}{Circle} and \ovalnode{B}{Oval}
\ncbar[angle=90]{A}{B}
```

\dianode*[*par*]**{*name*}{*stuff*}**

This is like **\diabox**.

\trinode*[*par*]**{*name*}{*stuff*}**

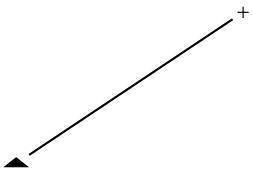
This is like **\tribox**.




```
\rput[tl](0,3){\dianode{A}{Diamond}}
\rput[br](4,0){\trinode[trimode=L]{B}{Triangle}}
\nccurve[angleA=-135,angleB=90]{A}{B}
```

\dotnode*[*par*](*x*,*y*)**{*name*}**

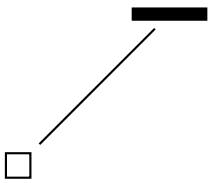
This is a variant of **\psdot**. For example:



```
\dotnode[dotstyle=triangle*,dotscale=2  1](0,0){A}
\dotnode[dotstyle=+](3,2){B}
\ncline[nodesep=3pt]{A}{B}
```

\fnode*[*par*](*x*,*y*)**{*name*}**

The f stands for “frame”. This is like, but easier than, putting a **\psframe** in an **\rnode**.



```
\fnode{A}
\fnode*[framesize=1  5pt](2,2){B}
\ncline[nodesep=3pt]{A}{B}
```

There are two differences between **\fnode** and **\psframe**:

- There is a single (optional) coordinate argument, that gives the *center* of the frame.
- The width and height of the frame are set by the

  framesize=*dim1 ‘dim2’* **Default: 10pt**

  parameter. If you omit *dim2*, you get a square frame.

7 Node connections

All the node connection commands begin with nc, and they all have the same syntax:[1]

[1]The node connections can be used with **\pscustom**. The beginning of the node connection is attached to the current point by a straight line, as with **\psarc**.[2]

nodeconnection[*par*]{*arrows*}{*nodeA*}{*nodeB*}

A line of some sort is drawn from *nodeA* to *nodeB*. Some of the node connection commands are a little confusing, but with a little experimentation you will figure them out, and you will be amazed at the things you can do. When we refer to the A and B nodes below, we are referring only to the order in which the names are given as arguments to the node connection macros.[3]

The node connections use many of the usual graphics parameters, plus a few special ones. Let's start with one that applies to all the node connections:

nodesep=*dim* **Default: 0pt**

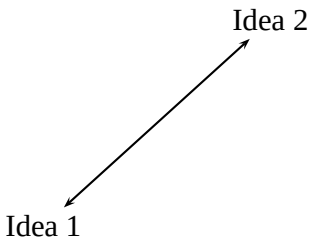
nodesep is the border around the nodes that is added for the purpose of determining where to connect the lines.

For this and other node connection parameters, you can set different values for the two ends of the node connection. Set the parameter **nodesepA** for the first node, and set **nodesepB** for the second node.

The first two node connections draw a line or arc directly between the two nodes:

\ncline*[*par*]{*arrows*}**{*nodeA*}{*nodeB*}**

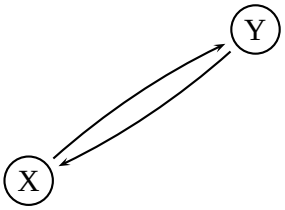
This draws a straight line between the nodes. For example:




```
\rput[bl](0,0){\rnode{A}{Idea 1}}
\rput[tr](4,3){\rnode{B}{Idea 2}}
\ncline[nodesep=3pt]{<->}{A}{B}
```

\ncarc*[*par*]{*arrows*}**{*nodeA*}{*nodeB*}**

This connects the two nodes with an arc.




```
\cnodeput(0,0){A}{X}
\cnodeput(3,2){B}{Y}
\psset{nodesep=3pt}
\ncarc{->}{A}{B}
\ncarc{->}{B}{A}
```

[3] When a node name cannot be found on the same page as the node connection command, you get either no node connection or a nonsense node connection. However, TEX will not report any errors.

The angle between the arc and the line between the two nodes is[4]

arcangle=*angle* **Default: 8**

\ncline and **\ncarc** both determine the angle at which the node connections join by the relative position of the two nodes. With the next group of node connections, you specify one or both of the angles in absolute terms, by setting the

angle=*angle* **Default: 0**

(and **angleA** and **angleB**) parameter.

You also specify the length of the line segment where the node connection joins at one or both of the ends (the "arms") by setting the

arm=*dim* **Default: 10pt**

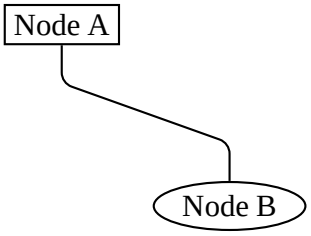
(and **armA** and **armB**) parameter.

These node connections all consist of several line segments, including the arms. The value of **linearc** is used for rounding the corners.

Here they are, starting with the simplest one:

\ncdiag*[*par*]{*arrows*}{*nodeA*}{*nodeB*}

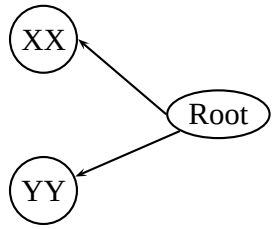
An arm is drawn at each node, joining at angle **angleA** or **angleB**, and with a length of **armA** or **armB**. Then the two arms are connected by a straight line, so that the whole line has three line segments. For example:




```
\rput[tl](0,3){\rnode{A}{\psframebox{Node A}}}
\rput[br](4,0){\ovalnode{B}{Node B}}
\ncdiag[angleA=-90, angleB=90, arm=.5, linearc=.2]{A}{B}
```

[4]Rather than using a true arc, **\ncarc** actually draws a bezier curve. When connecting two circular nodes using the default parameter values, the curve will be indistinguishable from a true arc. However, **\ncarc** is more flexible than an arc, and works right connecting nodes of different shapes and sizes. You can set **arcangleA** and **arcangleB** separately, and you can control the curvature with the **ncurv** parameter, which is described on page **??**.

You can also set one or both of the arms to zero length. For example, if you set **arm=0**, the nodes are connected by a straight line, but you get to determine where the line connects (whereas the connection point is determined automatically by **\ncline**). Compare this use of **\ncdiag** with **\ncline** in the following example:

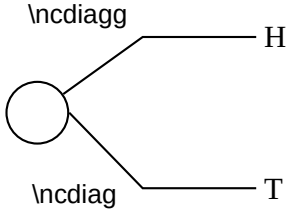



```
\rput[r](4,1){\ovalnode{R}{Root}}
\cnodeput(1,2){A}{XX}
\cnodeput(1,0){B}{YY}
\ncdiag[angleB=180, arm=0]{<-}{A}{R}
\ncline{<-}{B}{R}
```

(Note that in this example, the default value **angleA=0** is used.)

\ncdiagg*[*par*]{*arrows*}**{*nodeA*}{*nodeB*}**

\ncdiagg is similar to **\ncdiag**, but only the arm for node A is drawn. The end of this arm is then connected directly to node B. Compare **\ncdiagg** with **\ncdiag** when **armB=0**:

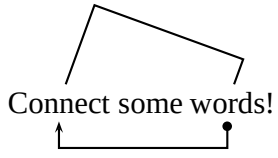



```
\cnode(0,0){12pt}{a}
\rput[l](3,1){\rnode{b}{H}}
\rput[l](3,-1){\rnode{c}{T}}
\ncdiagg[angleA=180, armA=1.5, nodesepA=3pt]{b}{a}
\ncdiag[angleA=180, armA=1.5, armB=0, nodesepA=3pt]{c}{a}
```

You can use **\ncdiagg** with **armA=0** if you want a straight line that joins to node A at the angle you specify, and to node B at an angle that is determined automatically.

\ncbar*[*par*]{*arrows*}**{*nodeA*}{*nodeB*}**

This node connection consists of a line with arms dropping “down”, at right angles, to meet two nodes at an angle **angleA**. Each arm is at least of length **armA** or **armB**, but one may be need to be longer.

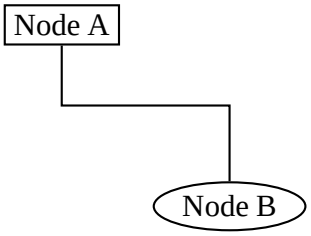



```
\rnode{A}{Connect} some \rnode{B}{words}!
\ncbar[nodesep=3pt,angle=-90]{<-**}{A}{B}
\ncbar[nodesep=3pt,angle=70]{A}{B}
```

Generally, the whole line has three straight segments.

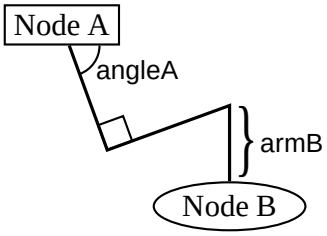
\ncangle*[*par*]{*arrows*}**{*nodeA*}{*nodeB*}**

Now we get to a more complicated node connection. **\ncangle** typically draws three line segments, like **\ncdiag**. However, rather than fixing the length of arm A, we adjust arm A so that the line joining the two arms meets arm A at a right angle. For example:



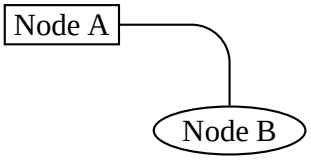

```
\rput[tl](0,3){\rnode{A}{\psframebox{Node A}}}
\rput[br](4,0){\ovalnode{B}{Node B}}
\ncangle[angleA=-90,angleB=90,armB=1cm]{A}{B}
```

Now watch what happens when we change **angleA**:




```
\rput[tl](0,3){\rnode{A}{\psframebox{Node A}}}
\rput[br](4,0){\ovalnode{B}{Node B}}
\ncangle[angleA=-70,angleB=90,armB=1cm,linewidth=1.2pt]{A}{B}
```

\ncangle is also a good way to join nodes by a right angle, with just two line segments, as in this example:

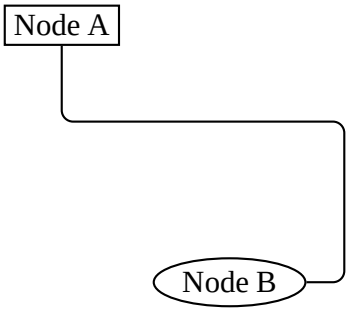



```
\rput[tl](0,2){\rnode{A}{\psframebox{Node A}}}
\rput[br](4,0){\ovalnode{B}{Node B}}
\ncangle[angleB=90, armB=0, linearc=.5]{A}{B}
```

\ncangles*[*par*]{*arrows*}**{*nodeA*}{*nodeB*}**

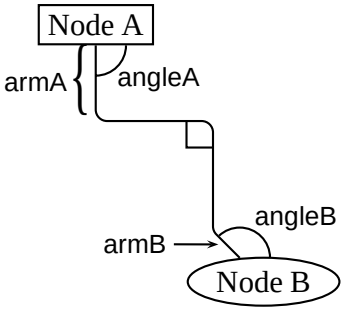
\ncangles is similar to **\ncangle**, but the length of arm A is fixed by the **armA** parameter. Arm A is connected to arm B by two line segments that meet arm A and each other at right angles. The angle at which they join arm B, and the length of the connecting segments, depends on the positions of the two arms. **\ncangles** generally draws a total of four line segments.[5] For example:

[5] Hence there is one more angle than **\ncangle**, and hence the s in **\ncangles**.




```
\rput[tl](0,4){\rnode{A}{\psframebox{Node A}}}
\rput[br](4,0){\ovalnode{B}{Node B}}
\ncangles[angleA=-90, armA=1cm, armB=.5cm, linearc=.15]{A}{B}
```

Let's see what happens to the previous example when we change **angleB**:



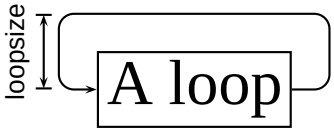

```
\rput[tl](0,4){\rnode{A}{\psframebox{Node A}}}
\rput[br](4,0){\ovalnode{B}{Node B}}
\ncangles[angleA=-90, angleB=135, armA=1cm, armB=.5cm,
    linearc=.15]{A}{B}
```

\ncloop*[*par*]{*arrows*}**{*nodeA*}{*nodeB*}**

\ncloop is also in the same family as **\ncangle** and **\ncangles**, but now typically 5 line segments are drawn. Hence, **\ncloop** can reach around to opposite sides of the nodes. The lengths of the arms are fixed by **armA** and **armB**. Starting at arm A, **\ncloop** makes a 90 degree turn to the left, drawing a segment of length

loopsize=*dim* **Default: 1cm**

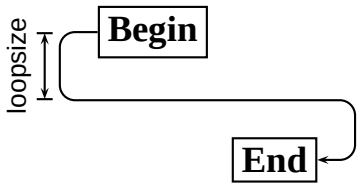
This segment connects to arm B the way arm A connects to arm B with **\ncline**; that is, two more segments are drawn, which join the first segment and each other at right angles, and then join arm B. For example:




```
\rnode{a}{\psframebox{\Huge A loop}}
\ncloop[angleB=180,loopsize=1,arm=.5,linearc=.2]{->}{a}{a}
```

In this example, node A and node B are the same node! You can do this with all the node connections (but it doesn't always make sense).

Here is an example where **\ncloop** connects two different nodes:

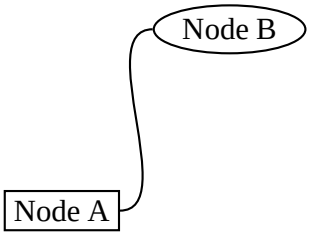



```
\parbox{3cm}{%
\rnode{A}{\psframebox{\large\bf  Begin}}
\vspace{1cm}\hspace*{\fill}
\rnode{B}{\psframebox{\large\bf  End}}
\ncloop[angleA=180,loopsize=.9,arm=.5,linearc=.2]{->}{A}{B}}
```

The next two node connections are a little different from the rest.

\nccurve*[*par*]{*arrows*}**{*nodeA*}{*nodeB*}**

\nccurve draws a bezier curve between the nodes.




```
\rput[bl](0,0){\rnode{A}{\psframebox{Node  A}}}
\rput[tr](4,3){\ovalnode{B}{Node  B}}
\nccurve[angleB=180]{A}{B}
```

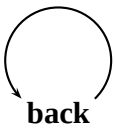
You specify the angle at which the curve joins the nodes by setting the **angle** (and **angleA** and **angleB**) parameter. The distance to the control points is set with the

ncurv=*num* **Default: .67**

(and **ncurvA** and **ncurvB**) parameter. A lower number gives a tighter curve. (The distance between the beginning of the arc and the first control point is one-half **ncurvA** times the distance between the two endpoints.)

\nccircle*[*par*]{*arrows*}**{*node*}{*radius*}**

\nccircle draws a circle, or part of a circle, that, if complete, would pass through the center of the node counterclockwise, at an angle of **angleA**.




```
\rnode{A}{\bf  back}
\nccircle[nodesep=3pt]{->}{A}{.7cm}
\kern  5pt
```

\nccircle can only connect a node to itself; it is the only node connection with this property. **\nccircle** is also special because it has an additional argument, for specifying the radius of the circle.

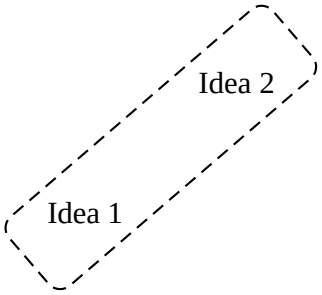
The last two node connections are also special. Rather than connecting the nodes with an open curve, they enclose the nodes in a box or curved box. You can think of them as variants of **\ncline** and **\ncarc**. In both cases, the half the width of the box is

boxsize=*dim* **Default: .4cm**

You have to set this yourself to the right size, so that the nodes fit inside the box. The **boxsize** parameter actually sets the **boxheight** and **boxdepth** parameters. The ends of the boxes extend beyond the nodes by **nodesepA** and **nodesepB**.

\ncbox*[*par*]{*nodeA*}{*nodeB*}

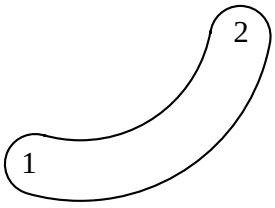
\ncbox encloses the nodes in a box with straight sides. For example:




```
\rput[bl](.5,0){\rnode{A}{Idea 1}}
\rput[tr](3.5,2){\rnode{B}{Idea 2}}
\ncbox[nodesep=.5cm,boxsize=.6,linearc=.2,
   linestyle=dashed]{A}{B}
```

\ncarcbox*[*par*]{*nodeA*}{*nodeB*}

\ncarcbox encloses the nodes in a curved box that is **arcangleA** away from the line connecting the two nodes.




```
\rput[bl](.5,0){\rnode{A}{1}}
\rput[tr](3.5,2){\rnode{B}{2}}
\ncarcbox[nodesep=.2cm,boxsize=.4,linearc=.4,
   arcangle=50]{<->}{A}{B}
```

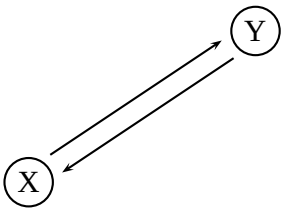
The arc is drawn counterclockwise from node A to node B.

There is one other node connection parameter that applies to all the node connections, except **\ncarcbox**:

offset=*dim* **Default: 0pt**

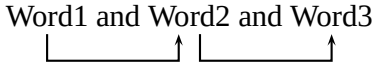
(You can also set **offsetA** and **offsetB** independently.) This shifts the point where the connection joins up by *dim* (given the convention that connections go from left to right).

There are two main uses for this parameter. First, it lets you make two parallel lines with **\ncline**, as in the following example:



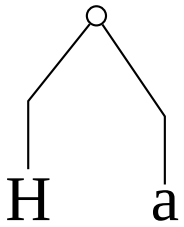

```
\cnodeput(0,0){A}{X}
\cnodeput(3,2){B}{Y}
\psset{nodesep=3pt,offset=4pt,arrows=->}
\ncline{A}{B}
\ncline{B}{A}
```

Second, it lets you join a node connection to a rectangular node at a right angle, without limiting yourself to positions that lie directly above, below, or to either side of the center of the node. This is useful, for example, if you are making several connections to the same node, as in the following example:



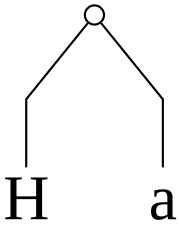

```
\rnode{A}{Word1} and \rnode{B}{Word2} and \rnode{C}{Word3}
\ncbar[offsetB=4pt,angleA=-90,nodesep=3pt]{->}{A}{B}
\ncbar[offsetA=4pt,angleA=-90,nodesep=3pt]{->}{B}{C}
```

Sometimes you might be aligning several nodes, such as in a tree, and you want to ends or the arms of the node connections to line up. This won't happen naturally if the nodes are of different size, as you can see in this example:




```
\Huge
\cnode(1,3){4pt}{a}
\rput[B](0,0){\Rnode{b}{H}}
\rput[B](2,0){\Rnode{c}{a}}
\psset{angleA=90,armA=1,nodesepA=3pt}
\ncdiagg{b}{a}
\ncdiagg{c}{a}
```

If you set the **nodesep** or **arm** parameter to a negative value, PSTricks will measure the distance to the beginning of the node connection or to the end of the arm relative to the center of the node, rather than relative to the boundary of the node or the beginning of the arm. Here is how we fix the previous example:




```
\Huge
\cnode(1,3){4pt}{a}
\rput[B](0,0){\Rnode{b}{H}}
\rput[B](2,0){\Rnode{c}{a}}
\psset{angleA=90,armA=1,nodesepA=-12pt}
\ncdiagg{b}{a}
\ncdiagg{c}{a}
```

Note also the use of **\Rnode**.

One more parameter trick: By using the **border** parameter, you can create the impression that one node connection passes over another.

The node connection commands make interesting drawing tools as well, as an alternative to **\psline** for connecting two points. There are variants of the node connection commands for this purpose. Each begins with pc (for “point connection”) rather than nc. E.g.,

```
\pcarc{<->}(3,4)(6,9)
```

gives the same result as

```
\pnode(3,4){A}
\pnode(6,9){B}
\pcarc{<->}{A}{B}
```

Only **\nccircle** does not have a pc variant:

Command	*Corresponds to:*
\pcline{*arrows*}**(*x1*,*y1*)(*x2*,*y2*)**	**\ncline**
\pccurve{*arrows*}**(*x1*,*y1*)(*x2*,*y2*)**	**\nccurve**
\pcarc{*arrows*}**(*x1*,*y1*)(*x2*,*y2*)**	**\ncarc**
\pcbar{*arrows*}**(*x1*,*y1*)(*x2*,*y2*)**	**\ncbar**
\pcdiag{*arrows*}**(*x1*,*y1*)(*x2*,*y2*)**	**\ncdiag**
\pcangle{*arrows*}**(*x1*,*y1*)(*x2*,*y2*)**	**\ncangle**
\pcloop{*arrows*}**(*x1*,*y1*)(*x2*,*y2*)**	**\ncloop**
\pcbox(*x1*,*y1*)(*x2*,*y2*)	**\ncbox**
\pcarcbox(*x1*,*y1*)(*x2*,*y2*)	**\ncarcbox**

8 Node connections labels: I

Now we come to the commands for attaching labels to the node connections. The label command must come right after the node connection to

which the label is to be attached. You can attach more than one label to a node connection, and a label can include more nodes.

The node label commands must end up on the same TEX page as the node connection to which the label corresponds.

There are two groups of connection labels, which differ in how they select the point on the node connection. In this section we describe the first group:

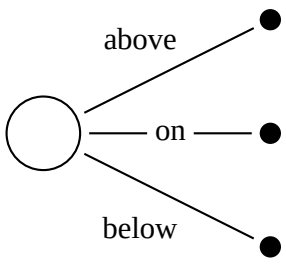
\ncput*[*par*]**{*stuff*}**
\naput*[*par*]**{*stuff*}**
\nbput*[*par*]**{*stuff*}**

These three command differ in where the labels end up with respect to the line:

\ncput	*on* the line
\naput	*above* the line
\nbput	*below* the line

(using the convention that node connections go from left to right).

Here is an example:




```
\cnode(0,0){.5cm}{root}
\cnode*(3,1.5){4pt}{A}
\cnode*(3,0){4pt}{B}
\cnode*(3,-1.5){4pt}{C}
\psset{nodesep=3pt}
\ncline{root}{A}
\naput{above}
\ncline{root}{B}
\ncput*{on}
\ncline{root}{C}
\nbput{below}
```

\naput and **\nbput** use the same algorithm as **\uput** for displacing the labels, and the distance beteen the line and labels is **labelsep** (at least if the lines are straight).

\ncput uses the same system as **\rput** for setting the reference point. You change the reference point by setting the

ref=*ref* **Default: c**

parameter.

Rotation is also controlled by a graphics parameter:

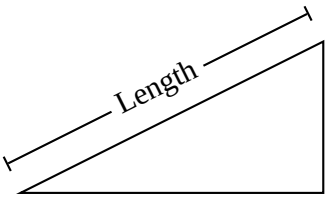
nrot=*rot* **Default: 0**

rot can be in any of the forms suitable for **\rput**, and you can also use the form

{:*angle*}

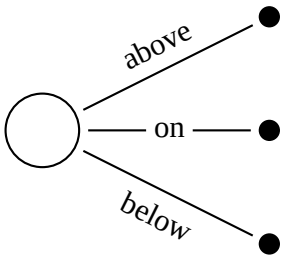
The angle is then measured with respect to the node connection. E.g., if the angle is {:U}, then the label runs parallel to the node connection. Since the label can include other put commands, you really have a lot of control over the label position.

The next example illustrates the use {:*angle*}, the **offset** parameter, and **\pcline**:




```
\pspolygon(0,0)(4,2)(4,0)
\pcline[offset=12pt]{|-|}(0,0)(4,2)
\ncput*[nrot=:U]{Length}
```

Here is a repeat of an earlier example, now using {:*angle*}:




```
\cnode(0,0){.5cm}{root}
\cnode*(3,1.5){4pt}{A}
\cnode*(3,0){4pt}{B}
\cnode*(3,-1.5){4pt}{C}
\psset{nodesep=3pt,nrot=:U}
\ncline{root}{A}
\naput{above}
\ncline{root}{B}
\ncput*{on}
\ncline{root}{C}
\nbput{below}
```

The position on the node connection is set by the

npos=*num* **Default:**

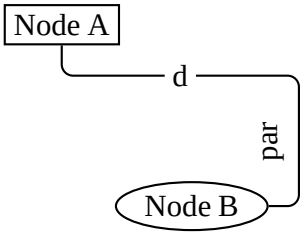
parameter, roughly according to the following scheme: Each node connection has potentially one or more segments, including the arms and connecting lines. A number **npos** between 0 and 1 picks a point on the first segment from node A to B (fraction **npos** from the beginning to the end of the segment), a number between 1 and 2 picks a number on the second segment, and so on.

Each node connection has its own default value of **npos**. If you leave the **npos** parameter value empty (e.g., [npos=]), then the default is substituted. This is the default mode.

Here are the details for each node connection:

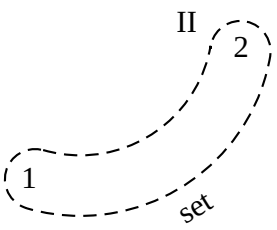
Connection	*Segments*	*Range*	*Default*
\ncline	1	$0 \le pos \le 1$	0.5
\nccurve	1	$0 \le pos \le 1$	0.5
\ncarc	1	$0 \le pos \le 1$	0.5
\ncbar	3	$0 \le pos \le 3$	1.5
\ncdiag	3	$0 \le pos \le 3$	1.5
\ncdiagg	2	$0 \le pos \le 2$	0.5
\ncangle	3	$0 \le pos \le 3$	1.5
\ncangles	4	$0 \le pos \le 4$	1.5
\ncloop	5	$0 \le pos \le 5$	2.5
\nccircle	1	$0 \le pos \le 1$	0.5
\ncbox	4	$0 \le pos \le 4$	0.5
\ncarcbox	4	$0 \le pos \le 4$	0.5

Here is an example:




```
\rput[tl](0,3){\rnode{A}{\psframebox{Node  A}}}
\rput[br](3.5,0){\ovalnode{B}{Node  B}}
\ncangles[angleA=-90,arm=.4cm,linearc=.15]{A}{B}
\ncput*{d}
\nbput[nrot=:D,npos=2.5]{par}
```

With **\ncbox** and **\ncarcbox**, the segments run counterclockwise, starting with the lower side of the box. Hence, with **\nbput** the label ends up outside the box, and with **\naput** the label ends up inside the box.

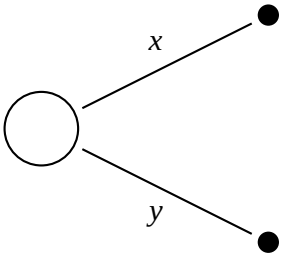



```
\rput[bl](.5,0){\rnode{A}{1}}
\rput[tr](3.5,2){\rnode{B}{2}}
\ncarcbox[nodesep=.2cm,boxsize=.4,linearc=.4,
   arcangle=50,linestyle=dashed]{<->}{A}{B}
\nbput[nrot=:U]{set}
\nbput[npos=2]{II}
```

If you set the parameter

shortput=*none/nab/tablr/tab* **Default: none**

to nab, then immediately following a node connection or another node connection label you can use ˆ instead of **\naput** and _ instead of **\nbput**.




```
\cnode(0,0){.5cm}{root}
\cnode*(3,1.5){4pt}{A}
\cnode*(3,-1.5){4pt}{C}
\psset{nodesep=3pt,shortput=nab}
\ncline{root}{A}^{$x$}
\ncline{root}{C}_{$y$}
```

You can still have parameter changes with the short ˆ and _ forms. Another example is given on page 27.

If you have set **shortput=nab**, and then you want to use a true ˆ or _ character right after a node connection, you must precede the ˆ or _ by {} so that PSTricks does not convert it to \naput or \nbput.

You can change the characters that you use for the short form with the

\MakeShortNab{*char1*}{*char2*}

command.[6]

The **shortput=tablr** and **shortput=tab** options are described on pages 24 and **??**, respectively.

9 Node connection labels: II

Now the second group of node connections:

[6] You can also use **\MakeShortNab** if you want to use ˆ and _ with non-standard category codes. Just invoke the command after you have made your \catcode changes.

\tvput*[*par*]**{*stuff*}**
\tlput*[*par*]**{*stuff*}**
\trput*[*par*]**{*stuff*}**
\thput*[*par*]**{*stuff*}**
\taput*[*par*]**{*stuff*}**
\tbput*[*par*]**{*stuff*}**

The difference between these commands and the \n*put commands is that these find the position as an intermediate point between the centers of the nodes, either in the horizontal or vertical direction. These are good for trees and mathematical diagrams, where it can sometimes be nice to have the labels be horizontally or vertically aligned. The t stands for “tree”.

You specify the position by setting the

tpos=*num* **Default: .5**

parameter.

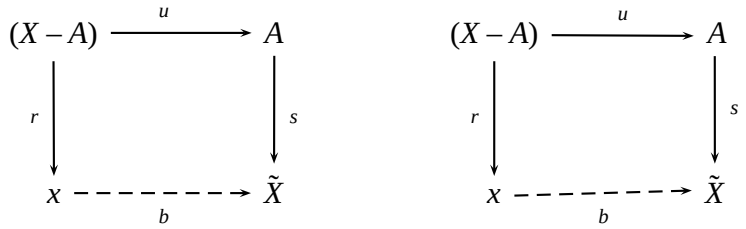
\tvput, **\tlput** and **\trput** find the position that lies fraction *tpos* in the *vertical* direction from the upper node to the lower node. **\thput**, **\taput** and **\tbput** find the position that lies fraction *tpos* in the *horizontal* direction from the left node to the right node. Then the commands put the label on or next to the line, as follows:

Command	*Direction*	*Placement*
\tvput	vertical	middle
\tlput	vertical	left
\trput	vertical	right
\thput	horizontal	middle
\taput	horizontal	above
\tbput	horizontal	below

Here is an example:

```
\[
  \setlength{\arraycolsep}{1.1cm}
  \begin{array}{cc}
    \Rnode{a}{(X-A)} & \Rnode{b}{A} \\[1.5cm]
    \Rnode{c}{x} & \Rnode{d}{\tilde{X}}
  \end{array}
```

```
    \psset{nodesep=5pt,arrows=->}
    \everypsbox{\scriptstyle}
    \ncline{a}{c}\tlput{r}
    \ncline{a}{b}\taput{u}
    \ncline[linestyle=dashed]{c}{d}\tbput{b}
    \ncline{b}{d}\trput{s}
\]
```



On the left is the diagram with **\tlput**, **\trput** **\tbput** and **\Rnode**, as shown in the code. On the right is the same diagram, but with **\naput** **\nbput** and **\rnode**.

These do not have a rotation argument or parameter. However, you can rotate *stuff* in 90 degree increments using box rotations (e.g., **\rotateleft**).

If you set **shortput=tablr**, then you can use the following single-character abbreviations for the t put commands:

Char.	*Short for:*
^	**\taput**
_	**\tbput**
<	**\tlput**
>	**\trput**

You can change the character abbreviations with

\MakeShortTablr{*char1*}{*char2*}{*char3*}{*char4*}

The t put commands, including an example of **shortput=tablr**, will be shown further when we get to mathematical diagrams and trees.

Driver notes: The node macros use **\pstVerb** and **\pstverbscale**.

10 Attaching labels to nodes

The command

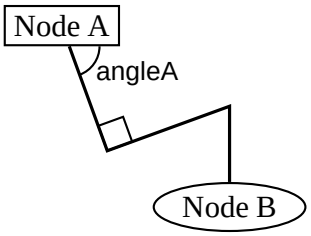
\nput*[*par*]**{*refangle*}{*name*}{*stuff*}**

affixes *stuff* to node *name*. It is positioned distance **labelsep** from the node, in the direction *refangle* from the center of the node. The algorithm is the same as for **\uput**. If you want to rotate the node, set the

rot=*rot* **Default: 0**

parameter, where *rot* is a rotation that would be valid for **\rput**.[7] The position of the label also takes into account the **offsetA** parameter. If **labelsep** is negative, then the distance is from the center of the node rather than from the boundary, as with **nodesep**.

Here is how I used **\nput** to mark an angle in a previous example:




```
\rput[br](4,0){\ovalnode{B}{Node B}}
\rput[tl](0,3){\rnode{A}{\psframebox{Node A}}}
\nput[labelsep=0]{-70}{A}{%
   \psarcn(0,0){.4cm}{0}{-70}
   \uput{.4cm}[-35](0,0){{\tt angleA}}}
\ncangle[angleA=-70,angleB=90,armB=1cm,linewidth=1.2pt]{A}{B}
\ncput[nrot=:U,npos=1]{\psframe[dimen=middle](0,0)(.35,.35)}
```

11 Mathematical diagrams and graphs

For some applications, such as mathematical diagrams and graphs, it is useful to arrange nodes on a grid. You can do this with alignment environments, such as TEX's \halignprimitive, LATEX's tabular environment, and AMS-TEX's \matrix, but PSTricks contains its own alignment environment that is especially adapted for this purpose:

\psmatrix ... \endpsmatrix

Here is an example

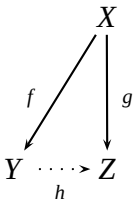
$$
\begin{array}{ccc}
 & A & \\
B & E & C \\
 & D &
\end{array}
$$

```
$
\psmatrix[colsep=1cm,rowsep=1cm]
     & A \\
  B  & E & C \\
     & D &
\endpsmatrix
$
```

[7] Not to be confused with the nput parameter.

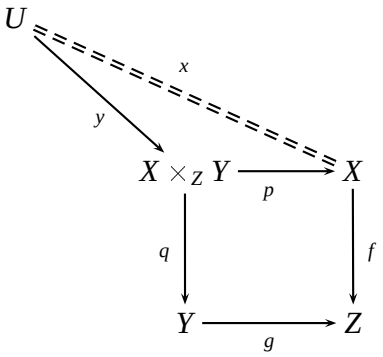
As an alignment environment, **\psmatrix** is similar to AMS-TEX's \matrix. There is no argument for specifying the columns. Instead, you can just use as many columns as you need. The entries are horizontally centered. Rows are ended by \\. **\psmatrix** can be used in or out of math mode.

Our first example wasn't very interesting, because we didn't make use of the nodes. Actually, each entry is a node. The name of the node in row *row* and column *col* is {*row*,*col*}, with no spaces. Let's see some node connections:




```
$
\psmatrix[colsep=1cm]
      & X \\
  Y & Z
\endpsmatrix
\everypsbox{\scriptstyle}%
\psset{nodesep=3pt,arrows=->}
\ncline{1,2}{2,1}
\tlput{f}
\ncline{1,2}{2,2}
\trput{g}
\ncline[linestyle=dotted]{2,1}{2,2}
\tbput{h}
$
```

You can include the node connections inside the **\psmatrix**, in the last entry and right before **\endpsmatrix**. One advantage to doing this is that **shortput=tablr** is the default within a **\psmatrix**.



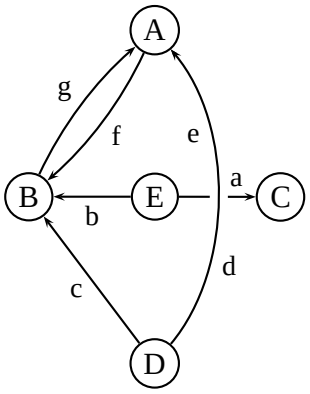

```
$
  \begin{psmatrix}
    U \\
      &   X\times_Z  Y  &  X \\
      &          Y          &  Z
    \psset{arrows=->,nodesep=3pt}
    \everypsbox{\scriptstyle}
    \ncline{1,1}{2,2}_{y}
    \ncline[doubleline=true,linestyle=dashed]{-}{1,1}{2,3}^{x}
    \ncline{2,2}{3,2}<{q}
    \ncline{2,2}{2,3}_{p}
    \ncline{2,3}{3,3}>{f}
    \ncline{3,2}{3,3}_{g}
  \end{psmatrix}
$
```

You can change the kind of nodes that are made by setting the

mnode=*type* **Default: R**

parameter. Valid types are R, r, C, f, p, circle, oval, dia, tri, dot and none, standing for **\Rnode**, **\rnode**, **\Cnode**, **\fnode**, **\pnode**, **\circlenode**, **\ovalnode**, **\dotnode** and no node, respectively. Note that for circles, you use **mnode=C** and set the radius with the **radius** parameter.

For example:



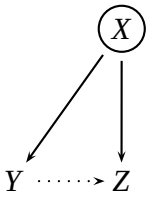

```
\psmatrix[mnode=circle,colsep=1]
      & A \\
   B  & E & C \\
      & D &
\endpsmatrix
\psset{shortput=nab,arrows=->,labelsep=3pt}
\small
\ncline{2,2}{2,3}^[npos=.75]{a}
\ncline{2,2}{2,1}^{b}
\ncline{3,2}{2,1}^{c}
\ncarc[arcangle=-40,border=3pt]{3,2}{1,2}
   _[npos=.3]{d}^[npos=.7]{e}
\ncarc[arcangle=12]{1,2}{2,1}^{f}
\ncarc[arcangle=12]{2,1}{1,2}^{g}
```

Note that a node is made only for the non-empty entries. You can also specify a node for the empty entries by setting the

emnode=*type* **Default: none**

parameter.

You can change parameters for a single entry by starting with entry with the parameter changes, enclosed in square brackets. Note that the changes affect the way the node is made, but not contents of the entry (use **\psset** for this purpose). For example:




```
$
\psmatrix[colsep=1cm]
      & [mnode=circle]  X  \\
   Y  &  Z
\endpsmatrix
\psset{nodesep=3pt,arrows=->}
\ncline{1,2}{2,1}
\ncline{1,2}{2,2}
\ncline[linestyle=dotted]{2,1}{2,2}
$
```

If you want your entry to begin with a [that is not meant to indicate parameter changes, then precede it by {}.

You can assign your own name to a node by setting the

name=*name* **Default:**

parameter at the beginning of the entry, as described above. You can still refer to the node by {*row*,*col*}, but here are a few reasons for giving your own name to a node:

- The name may be easier to keep track of;
- Unlike the {*row*,*col*} names, the names you give remain valid even when you add extra rows or columns to your matrix.
- The names remain valid even when you start a new **\psmatrix** that reuses the {*row*,*col*} names.

Here a few more things you should know:

- The baselines of the nodes pass through the centers of the nodes. **\psmatrix** achieves this by setting the

  nodealign=*true/false* **Default: false**

  parameter to true. You can also set this parameter outside of **\psmatrix** when you want this kind of alignment.

- You can left or right-justify the nodes by setting the

  mcol=*l/r/c* **Default: c**

  parameter. l, r and c stand for left, right and center, respectively.

- The space between rows and columns is set by the

rowsep=*dim* **Default: 1.5cm**
colsep=*dim* **Default: 1.5cm**

parameters.

- If you want all the nodes to have a fixed width, set

  mnodesize=*dim* **Default: -1pt**

  to a positive value.

- If **\psmatrix** is used in math mode, all the entries are set in math mode, but you can switch a single entry out of math mode by starting and ending the entry with `$`.

- The radius of the C **mnode** (corresponding to **\Cnode**) is set by the

  radius=*dim* **Default: 2pt**

  parameter.

- Like in LaTeX, you can end a row with `\\[`*dim*`]` to insert an extra space *dim* between rows.

- The command `\psrowhook`*ii* is executed, if defined, at the beginning of every entry in row *ii* (row 2), and the command `\pscolhook`*v* is executed at the beginning of every entry in column *v* (etc.). You can use these hooks, for example, to change the spacing between two columns, or to use a special **mnode** for all the entries in a particular row.

- An entry can itself be a node. You might do this if you want an entry to have two shapes.

- If you want an entry to stretch across several (*int*) columns, use the

  \psspan{*int*}

  at the end of the entry. This is like Plain TeX's **\multispan**, or LaTeX's **\multicolumn**, but the template for the current column (the first column that is spanned) is still used. If you want wipe out the template as well, use `\psspan{`*int}* *at the beginning of the entry* instead. If you just want to wipe out the template, use `\omit` before the entry.

- **\psmatrix** can be nested, but then all node connections and other references to the nodes in the {*row*,*col*} form for the nested matrix *must go inside* the **\psmatrix**. This is how PSTricks decides which matrix you are referring to. It is still neatest to put all the node connections towards the end; just be sure to put them before **\endpsmatrix**. Be careful also not to refer to a node until it actually appears. The whole matrix can itself go inside a node, and node connections can be made as usual. This is not the same as connecting nodes from two different **\psmatrix**'s. To do this, you must give the nodes names and refer to them by these names.

12 Obsolete put commands

This is old documentation, but these commands will continue to be supported.

There is also an obsolete command **\Lput** for putting labels next to node connections. The syntax is

\Lput{*labelsep*}[*refpoint*]{*rotation*}(*pos*){*stuff*}

It is a combination of **\Rput** and **\lput**, equivalent to

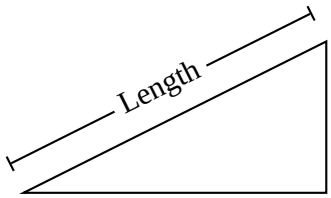
\lput(*pos*){\Rput{*labelsep*}[*refpoint*]{*rotation*}(0,0){*stuff*}}

\Mput is a short version of **\Lput** with no {*rotation*} or (*pos*) argument. **\Lput** and **\Mput** remain part of PSTricks only for backwards compatibility.

Here are the node label commands:

\lput*[*refpoint*]{*rotation*}***(pos){stuff}***

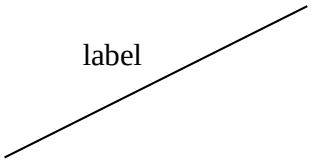
The l stands for "label". Here is an example illustrating the use of the optional star and :*angle* with **\lput**, as well as the use of the **offset** parameter with **\pcline**:




```
\pspolygon(0,0)(4,2)(4,0)
\pcline[offset=12pt]{|-|}(0,0)(4,2)
\lput*{:U}{Length}
```

(Remember that with the put commands, you can omit the coordinate if you include the angle of rotation. You are likely to use this feature with the node label commands.)

With **\lput** and **\rput**, you have a lot of control over the position of the label. E.g.,

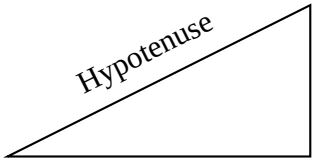



```
\pcline(0,0)(4,2)
\lput{:U}{\rput[r]{N}(0,.4){label}}
```

puts the label upright on the page, with right side located .4 centimeters "above" the position .5 of the node connection (above if the node connection points to the right). However, the **\aput** and **\bput** commands described below handle the most common cases without **\rput**.[8]

\aput*[*labelsep*]{*angle*}**(*pos*){*stuff*}**

stuff is positioned distance **\pslabelsep** *above* the node connection, given the convention that node connections point to the right. \aput is a node-connection variant of **\uput**. For example:




```
\pspolygon(0,0)(4,2)(4,0)
\pcline[linestyle=none](0,0)(4,2)
\aput{:U}{Hypotenuse}
```

\bput*[*labelsep*]{*angle*}**(*pos*){*stuff*}**

This is like **\aput**, but *stuff* is positioned *below* the node connection.

It is fairly common to want to use the default position and rotation with these node connections, but you have to include at least one of these arguments. Therefore, PSTricks contains some variants:

[8] There is also an obsolete command **\Lput** for putting labels next to node connections. The syntax is

\Lput{*labelsep*}[*refpoint*]{*rotation*}(*pos*){*stuff*}

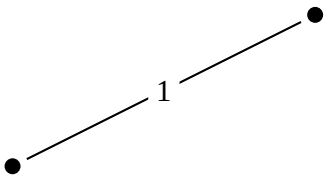
It is a combination of **\Rput** and **\lput**, equivalent to

\lput(*pos*){\Rput{*labelsep*}[*refpoint*]{*rotation*}(0,0){*stuff*}}

\Mput is a short version of **\Lput** with no {*rotation*} or (*pos*) argument. **\Lput** and **\Mput** remain part of PSTricks only for backwards compatibility.

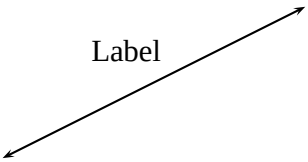
\mput*[*refpoint*]**{*stuff*}**
\Aput*[*labelsep*]**{*stuff*}**
\Bput*[*labelsep*]**{*stuff*}**

of **\lput**, **\aput** and **\bput**, respectively, that have no angle or positioning argument. For example:




```
\cnode*(0,0){3pt}{A}
\cnode*(4,2){3pt}{B}
\ncline[nodesep=3pt]{A}{B}
\mput*{1}
```

Here is another:



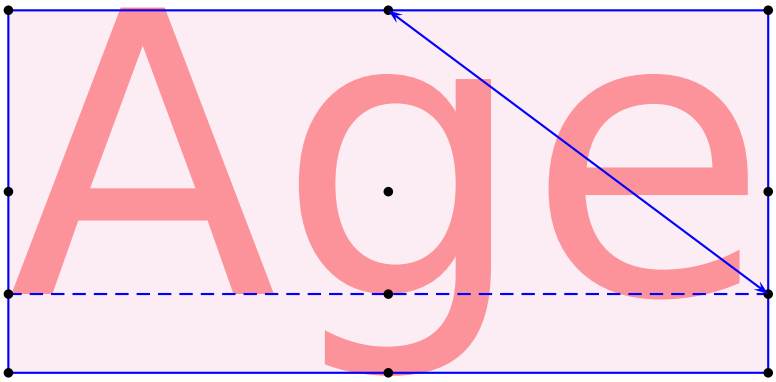

```
\pcline{<->}(0,0)(4,2)
\Aput{Label}
```

Part II.
New commands

1. Setting bounding box nodes with \psDefBoxNodes

\psDefBoxNodes{*node name*}

Setting nodes for a bounding box of a given text. There will be 12 nodes defined, with the name that has the suffixes :tl, :tC, :tr, :Cl, :C, :Cr, :Bl, :BC, :Br, :bl, :bc, :br. The prefix is always the given node name.

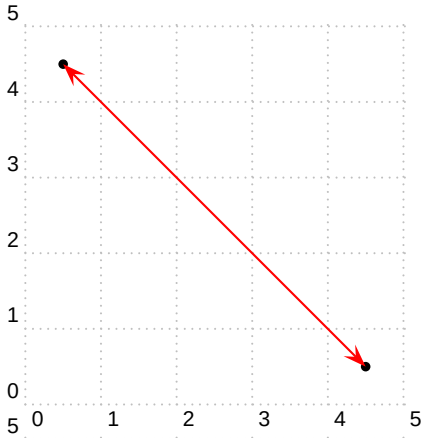



```
\psscalebox{15}{\psDefBoxNodes{Age}{\color{red!50}\sffamily Age}}%
\pspolygon[linecolor=blue,fillstyle=solid,
  fillcolor=magenta!30,opacity=0.3](Age:tl)(Age:tr)(Age:br)(Age:bl)%
\psline[linestyle=dashed,linecolor=blue](Age:Bl)(Age:Br)%
\psdots(Age:tl)(Age:tC)(Age:tr)(Age:Cl)(Age:C)(Age:Cr)%
     (Age:Bl)(Age:BC)(Age:Br)(Age:bl)(Age:bC)(Age:br)%
\pcline[arrows=<->,linecolor=blue,arrowscale=1.25](Age:tC)(Age:Br)
```

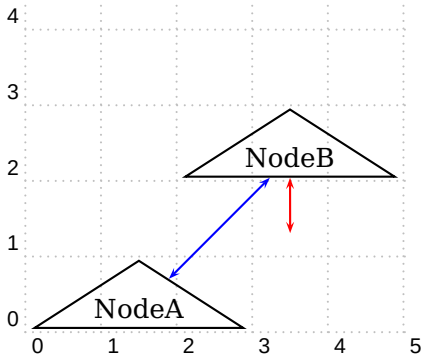
2. Relative nodes with \psGetNodeCenter

\psGetNodeCenter{*node name*}

This command makes sense only at the PostScript level. It defines the two variables *node.x* and *node.y* which can be used to define relative nodes. The following example defines the node MyNode and a second one relative to the first one, with 4 units left and 4 units up. *node* must be an existing node name.




```
\begin{pspicture}[showgrid=true,arrowscale=2](5,5)
\pnode(4.5,0.5){MyNode}
\psdot(MyNode)
\pnode(! \psGetNodeCenter{MyNode}
  MyNode.x 4 sub MyNode.y 4 add){MySecondNode}
\psdot(MySecondNode)
\ncline[linecolor=red]{<->}{MyNode}{MySecondNode}
\end{pspicture}
```

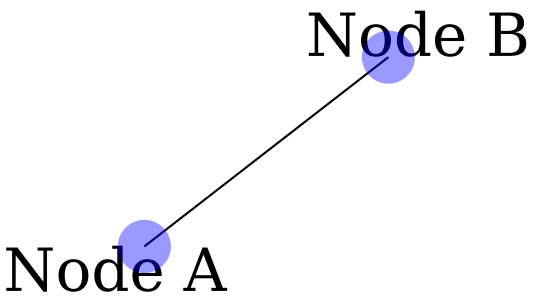



```
\begin{pspicture}[showgrid=true](5,5)
\rput(1.5,0.5){\trinode{CN}{NodeA}}
\rput(3.5,2.5){\trinode{EN}{NodeB}}
\pnode(! \psGetNodeCenter{CN}
  CN.x 2 add CN.y 1 add ){MyCNode}
\ncline[linecolor=red]{<->}{MyCNode}{EN}
\ncline[linecolor=blue]{<->}{CN}{EN}
\end{pspicture}
```

3. Getting node edges with \psGetNodeEdgeA and \psGetNodeEdgeB

\psGetNodeEdgeA{*node name*}
\psGetNodeEdgeB{*node name*}

When two nodes are connected the line often did not use the center of the defined nodes. Values as `nodesep` or `offset` are also taken into account as a surrounding border of a node. With these new macros one can get the edge coordinates of two given nodes. The coordinates are saved on `PostScript` side in the values *node.x* and *node.y*.



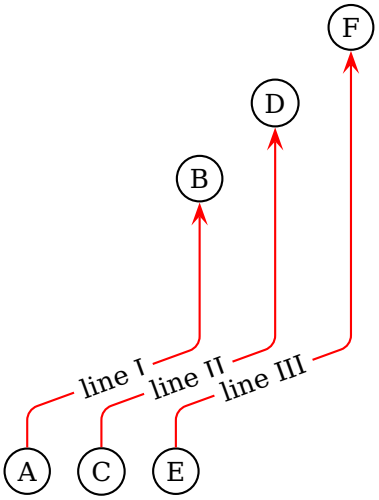

```
\Huge
\hspace*{4cm}\rnode{B}{Node B}

\vspace{2cm}
\rnode{A}{Node A}
\ncline{A}{B}
\pscircle*[linecolor=blue,opacity=0.4](!\psGetEdgeA{A}{B}){10pt}
\pscircle*[linecolor=blue,opacity=0.4](!\psGetEdgeB{A}{B}){10pt}
```

4. \ncdiag and \pcdiag

With the new option `lineAngle` the lines drawn by the `\ncdiag` macro can now have a specified gradient. Without this option one has to define the two arms (which maybe zero) and PSTricks draws the connection between them. Now there is only a static `armA`, the second one `armB` is calculated when an angle `lineAngle` is defined. This angle is the gradient of the intermediate line between the two arms. The syntax of `\ncdiag` is

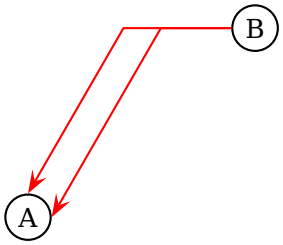
\ncdiag[Options]{*node A*}{*node B*}
\pcdiag[Options][*node A*][*node B*]

name	meaning
`lineAngle`	angle of the intermediate line segment. Default is 0, which is the same than using `\ncdiag` without the `lineAngle` option.

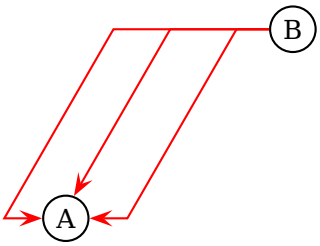



```
\begin{pspicture}(5,6)
  \circlenode{A}{A}\quad\circlenode{C}{C}%
    \quad\circlenode{E}{E}
  \rput(0,4){\circlenode{B}{B}}
  \rput(1,5){\circlenode{D}{D}}
  \rput(2,6){\circlenode{F}{F}}
  \psset{arrowscale=2,linearc=0.2,%
    linecolor=red,armA=0.5, angleA=90,angleB=-90}
  \ncdiag[lineAngle=20]{->}{A}{B}
  \ncput*[nrot=:U]{line I}
  \ncdiag[lineAngle=20]{->}{C}{D}
  \ncput*[nrot=:U]{line II}
  \ncdiag[lineAngle=20]{->}{E}{F}
  \ncput*[nrot=:U]{line III}
\end{pspicture}
```

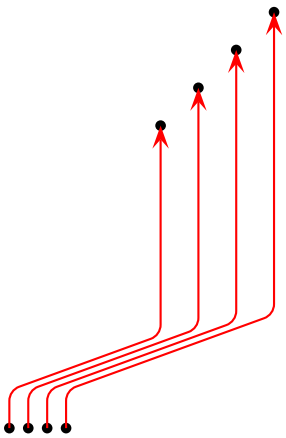
The \ncdiag macro sets the armB dynamically to the calculated value. Any user setting of armB is overwritten by the macro. The armA could be set to a zero length:




```
\begin{pspicture}(4,3)
  \rput(0.5,0.5){\circlenode{A}{A}}
  \rput(3.5,3){\circlenode{B}{B}}
  {\psset{linecolor=red,arrows=<-,arrowscale=2}
  \ncdiag[lineAngle=60,%
     armA=0,angleA=0,angleB=180]{A}{B}
  \ncdiag[lineAngle=60,%
     armA=0,angleA=90,angleB=180]{A}{B}}
\end{pspicture}
```




```
\begin{pspicture}(4,3)
  \rput(1,0.5){\circlenode{A}{A}}
  \rput(4,3){\circlenode{B}{B}}
  {\psset{linecolor=red,arrows=<-,arrowscale=2}
  \ncdiag[lineAngle=60,
     armA=0.5,angleA=0,angleB=180]{A}{B}
  \ncdiag[lineAngle=60,
     armA=0,angleA=70,angleB=180]{A}{B}
  \ncdiag[lineAngle=60,
     armA=0.5,angleA=180,angleB=180]{A}{B}}
\end{pspicture}
```

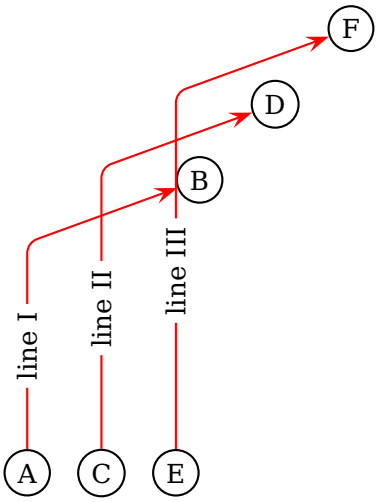


```
\begin{pspicture}(4,5.5)
  \cnode*(0,0){2pt}{A} \cnode*(0.25,0){2pt}{C}
  \cnode*(0.5,0){2pt}{E}\cnode*(0.75,0){2pt}{G}
  \cnode*(2,4){2pt}{B} \cnode*(2.5,4.5){2pt}{D}
  \cnode*(3,5){2pt}{F} \cnode*(3.5,5.5){2pt}{H}
  {\psset{arrowscale=2,linearc=0.2,
    linecolor=red,armA=0.5, angleA=90,angleB=-90}
  \pcdiag[lineAngle=20]{->}(A)(B)
  \pcdiag[lineAngle=20]{->}(C)(D)
  \pcdiag[lineAngle=20]{->}(E)(F)
  \pcdiag[lineAngle=20]{->}(G)(H)}
\end{pspicture}
```

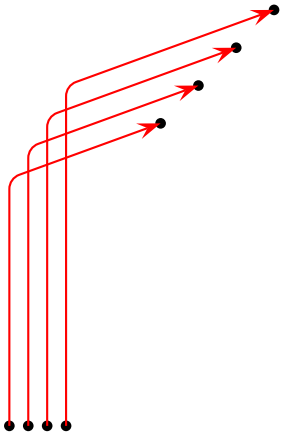
5. \ncdiagg and \pcdiagg

This is nearly the same as \ncdiag except that armB=0 and the angleB value is computed by the macro, so that the line ends at the node with an angle like a \pcdiagg line. The syntax of \ncdiagg/\pcdiagg is

```
\ncdiag [Options] {node A}{node B}
\pcdiag [Options] [node A][node B]
```



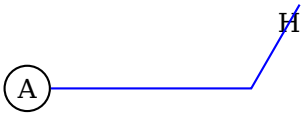

```
\begin{pspicture}(4,6)
  \psset{linecolor=black}
  \circlenode{A}{A}\quad\circlenode{C}{C}\quad%
    \circlenode{E}{E}
  \rput(0,4){\circlenode{B}{B}}
  \rput(1,5){\circlenode{D}{D}}
  \rput(2,6){\circlenode{F}{F}}
  {\psset{arrowscale=2,linearc=0.2,
    linecolor=red,armA=0.5, angleA=90}
  \ncdiagg[lineAngle=-160]{->}{A}{B}
  \ncput*[nrot=:U]{line I}
  \ncdiagg[lineAngle=-160]{->}{C}{D}
  \ncput*[nrot=:U]{line II}
  \ncdiagg[lineAngle=-160]{->}{E}{F}
  \ncput*[nrot=:U]{line III}}
\end{pspicture}
```



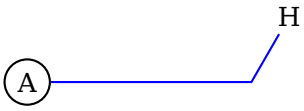
```
\begin{pspicture}(4,6)
  \psset{linecolor=black}
  \cnode*(0,0){2pt}{A} \cnode*(0.25,0){2pt}{C}
  \cnode*(0.5,0){2pt}{E}\cnode*(0.75,0){2pt}{G}
  \cnode*(2,4){2pt}{B} \cnode*(2.5,4.5){2pt}{D}
  \cnode*(3,5){2pt}{F} \cnode*(3.5,5.5){2pt}{H}
  {\psset{arrowscale=2,linearc=0.2,
    linecolor=red,armA=0.5, angleA=90}
  \pcdiagg[lineAngle=20]{->}(A)(B)
  \pcdiagg[lineAngle=20]{->}(C)(D)
  \pcdiagg[lineAngle=20]{->}(E)(F)
  \pcdiagg[lineAngle=20]{->}(G)(H)}
\end{pspicture}
```

The only catch for \ncdiagg is that you need the right value for lineAngle. If the node connection is on the wrong side of the second node, then choose the corresponding angle, e. g.: if 20 is wrong then take −160, which differs by 180.


```
\begin{pspicture}(4,1.5)
  \circlenode{a}{A}
  \rput[l](3,1){\rnode{b}{H}}
  \ncdiagg[lineAngle=60,angleA=180,armA=.5,nodesepA=3pt,
      linecolor=blue]{b}{a}
\end{pspicture}
```




```
\begin{pspicture}(4,1.5)
  \circlenode{a}{A}
  \rput[l](3,1){\rnode{b}{H}}
  \ncdiagg[lineAngle=60,armA=.5,nodesepB=3pt,linecolor=
      blue]{a}{b}
\end{pspicture}
```

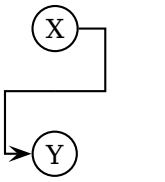



```
\begin{pspicture}(4,1.5)
  \circlenode{a}{A}
  \rput[l](3,1){\rnode{b}{H}}
  \ncdiagg[lineAngle=-120,armA=.5,nodesepB=3pt,linecolor
      =blue]{a}{b}
\end{pspicture}
```

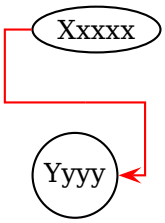
6. \ncbarr

This has the same behaviour as \ncbar, but has 5 segments and all are horizontal ones. This is the reason why angleA must be 0 or alternatively 180. All other values are set to 0 by the macro. The intermediate horizontal line is symmetrical to the distance of the two nodes.

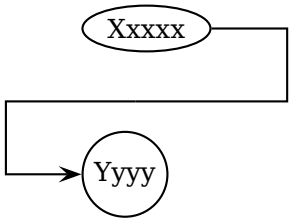
\ncbarr [Options] {*node A*}{*node B*}




```
\psset{arrowscale=2}%
\circlenode{X}{X}\\[1cm]
\circlenode{Y}{Y}
\ncbarr[angleA=0,arrows=->,arrowscale=2]{X}{Y}
```




```
\psset{arrowscale=2}%
\ovalnode{X}{Xxxxx}\\[1cm]
\circlenode{Y}{Yyyy}
\ncbarr[angleA=180,arrows=->,arrowscale=2,linecolor=red]{X
    }{Y}
```

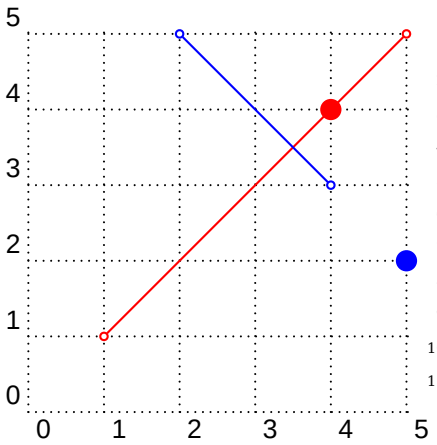



```
\psset{arrowscale=2}%
\ovalnode{X}{Xxxxx}\\[1cm]
\circlenode{Y}{Yyyy}
\ncbarr[angleA=20,arm=1cm,arrows=->,arrowscale=2]{X}{Y}
```

7. \psLNode and \psLCNode

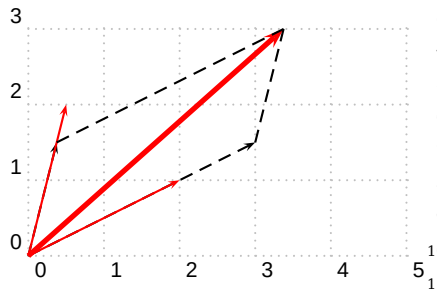
\psLNode interpolates the Line $\overline{AB}$ by the given value and sets a node at this point. The syntax is

\psLNode[*P1*][*P2*]{*value*}{*node name*}
\psLCNode[*P1*]{*value 1*}[*P2*]{*value 2*}{*node name*}



```
\begin{pspicture}(5,5)
\psgrid[subgriddiv=0,griddots=10]
\psset{linecolor=red}
\psline{o-o}(1,1)(5,5)
\psLNode(1,1)(5,5){0.75}{PI}
\qdisk(PI){4pt}
\psset{linecolor=blue}
\psline{o-o}(4,3)(2,5)
\psLNode(4,3)(2,5){-0.5}{PII}
\qdisk(PII){4pt}
\end{pspicture}
```

The \psLCNode macro builds the linear combination of the two given vectors and stores the end of the new vector as a node. All vectors start at $(0, 0)$, so a \rput maybe appropriate. The syntax is



```
\begin{pspicture}[showgrid=true](5,3)
\psset{linecolor=black}
\psline[linestyle=dashed]{->}(3,1.5)
\psline[linestyle=dashed]{->}(0.375,1.5)
\psset{linecolor=red}
\psline{->}(2,1)\psline{->}(0.5,2)
\psLCNode(2,1){1.5}(0.5,2){0.75}{PI}
\psline[linewidth=2pt]{->}(PI)
\psset{linecolor=black}
\psline[linestyle=dashed](3,1.5)(PI)
\psline[linestyle=dashed](0.375,1.5)(PI)
\end{pspicture}
```

8. \nlput and \psLDNode

\ncput allows you to set a label relative to the first node of the last node connection. With \nlput this can be done absolute to a given node. The syntax is different to the other node connection macros. It uses internally the macro \psLDNode which places a node absolute to two given points, starting from the first one.

```
\nlput [Options] (A)(B){distance}{text}
\psLDNode [Options] (A)(B){distance}{node name}
```

```
\begin{pspicture}(5,2)
\pnode(0,0){A}
\pnode(5,2){B}
\ncline{A}{B}
\psLDNode(A)(B){1.5cm}{KN}\qdisk(KN){2pt}
\nlput[nrot=:U](A)(B){1cm}{Test}
\nlput[nrot=:D](A)(B){2cm}{Test}
\nlput[nrot=:U](A)(B){3cm}{Test}
\nlput(A)(B){4cm}{Test}
\end{pspicture}
```

9. Extensions

9.1. Quick overview

All macros in this section are connected in one way or other with the construction or deployment of one or more nodes of type `\pnode`, which is to say in effect, named points. For the remainder of this section, node always means `\pnode`. Nodes are one of the most powerful features of pstricks—the “trickiest tricks” in the words of its originator. If used without appropriate caution, they can produce PostScript errors that can be difficult to track down. For example, suppose you have defined a node `A` by `\pnode(1,1){A}`, and then, a little later, you want to move the node a bit to the right, and you write `\pnode([nodesep=.5cm]A){A}`. On processing the file you will see an error message from ghostscript: `stackunderflow in --exch--`. The lesson is: you may not assign a node name if a node by that name is involved explicitly in its definition. To reassign a node name safely, you have to write instead something like

```
\pnode([nodesep=.5cm]A){Atemp}
\pnode(Atemp){A}
```

This problem afflicts a number of other node-forming macros based on a `\pnode` construction, as most are.

Nodes are more complicated than they appear. Each node is stored not only with a recipe for finding its coordinates, but also with the coordinate system in effect when it was defined. Part of the retrieval process involves modifying the coordinates if necessary so that they represent the same point on the page even if the coordinate system has changed. This is important, but has some unexpected consequences. Normally, the simplest way to translate an object is with `\rput`.

```
\pnode(1,1){P}% define P as (1,1)
\rput(2,3){\psdot{P}}% places dot at original P=(1,1)
```

is different from

```
\rput(2,3){\pnode(1,1){P}\psdot{P}}
% places dot at (1,1)+(2,3)
```

Effectively, `\rput` and `\uput` are not useful for translating previously defined nodes, but they are useful for defining new nodes relative to fixed positions.

The new macros in this section are of several kinds: (i) utility macros, some used internally by the package and some of general use; (ii) macros that manipulate one or more nodes to produce other nodes; (iii) constructions intended to be used with nodes and node sequences. By a node sequence is meant one or more nodes having a common root name followed by an index—eg, P0 P1 P2 ... P5 is a node sequence with root name P. It is easy to define such node sequences using the `\multido` macro, or using one of a number of macros in this section.

9.2. Node expressions

A number of macros in the package (eg, `\psxline`) permit the use of node expressions, by which is meant an expression like

```
.25(1,3)+.333(2;90)-1.2([nodesep=.5cm]Q)
```

which specifies a linear combination of points (the items enclosed in parentheses) specified in any manner acceptable to `\SpecialCoor`.

Node expressions are handled by `\nodexn`, which calls the macros `\hasparen` and `\parsenodexn` to do the real work. If you write code that needs to be able to handle node expressions, you use

`\nodexn{`*expr*`}{`*nodename*`}`

which returns a node `<nodename>` once `<expr>` has been fully parsed. It is safe to reuse a node name, as in

```
\nodexn{(P)+.5(1,2)}{P}
```

The following macros amount to special cases of node expressions.

`\AtoB(`*A*`)(`*B*`){`*C*`}`

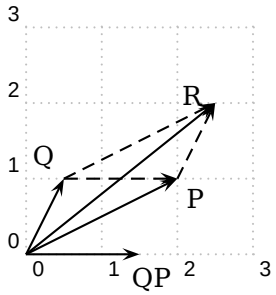
defines a node by name C essentially as B-A, as vectors. It is safe to use `\AtoB(Q)(P){P}` and `\AtoB(Q)(P){Q}`.

`\AplusB(`*A*`)(`*B*`){`*C*`}`

defines node by name C essentially as A+B, as vectors. It is safe to use `\AplusB(Q)(P){P}` and `\AplusB(Q)(P){Q}`.

`\midAB(`*A*`)(`*B*`){`*C*`}`

defines node by name C essentially as $(A+B)/2$, as vectors. It is safe to use `\midAB(Q)(P){P}` and `\midAB(Q)(P){Q}`.




```
\begin{pspicture}[showgrid=true](-.5,-.5)
    (2.5,2.5)
\psset{arrows=->,arrowscale=1.5}
\pnode(2,1){P}\pnode(.5,1){Q}
\AtoB(Q)(P){QP}
\AplusB(Q)(P){R}
\psline(0,0)(P)\uput[-45](P){P}
\psline(0,0)(Q)\uput[135](Q){Q}
\psline(0,0)(QP)\uput[-70](QP){QP}
\psline(0,0)(R)\uput[160](R){R}
\psline[linestyle=dashed](Q)(P)
\psline[linestyle=dashed](Q)(R)
\psline[linestyle=dashed](P)(R)
\end{pspicture}
```

9.3. The main macros

`\normalvec(`*coords*`){`*nodename*`}`

For example,

```
\normalvec(P){P}\normalvec(2;30){Q}
```

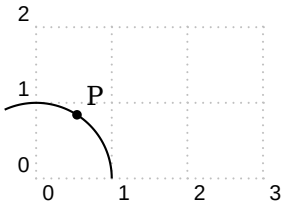
first redefines the node P as a node whose vector interpretation is of the same length as the original P, but rotated 90 degrees. The second instance has the same effect as \pnode(2;120){Q}.

\curvepnode{*tval*}{*expression in t*}{*nodename*}

For example,

```
\curvepnode{1}{cos(t) | sin(t)}{P}
```

sets a node named P at (cos(1), sin(1)) and a node named Ptang which represents a unit vector in the tangent direction to the curve at P. The expression in t in this case is algebraic, which is detected automatically by the macro.




```
\begin{pspicture}[showgrid=true](-.5,-.5)(2.5,2)
\def\exn{cos(t) | sin(t)}
\psparametricplot[algebraic]{0}{2}{\exn}
\curvepnode{1}{\exn}{P}
\psdot(P)\uput[45](P){P}
\end{pspicture}
```

\psparnode{*t*}{*expression in t*}{<*nodename*>}

is called by the command \curvename if the expression is PostScript, not algebraic.

\algparnode{*t*}{*expression in t*}{*nodename*}

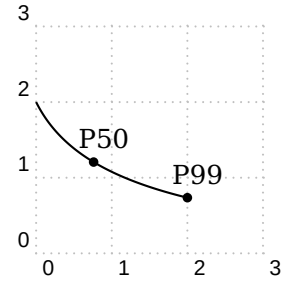
is called by the command \curvename if the expression is algebraic, not PostScript.

\curvepnodes{*tmin*}{*tmax*}{*expr. in t*}{*nodeRoot*}

Uses current setting of plotpoints (default 50) to define a node sequence of points along the curve. Eg,

```
\curvepnodes[plotpoints=100]{0}{1}{t+t^2 | Ex(-t)}{P}
```

sets nodes P0 .. P99 at equally spaced t values along the curve, and assigns the macro \Pnodecount to 99, the highest index. The expression in t may be either algebraic or PostScript, and is handled automatically. The values <tmin>, <tmax> may be expressed using PostScript—eg, {Pi neg}{PiDiv2}.




```
\begin{pspicture}[showgrid=true](-.5,-.5)(2.5,3)
\def\exn{t+t^2 | 2*Ex(-t)}
\psset{plotpoints=100}
\psparametricplot[algebraic]{0}{1}{\exn}
\curvepnodes{0}{1}{\exn}{P}
\psdot(P50)\uput[75](P50){P50}
\psdot(P99)\uput[75](P99){P99}
\end{pspicture}
```

`\fnpnode{`*xval*`}{`*expression in x*`}{`*nodename*`}`

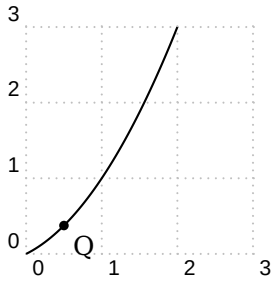
sets a single node on the graph. Eg,

```
\fnpnode{.5}{x x 1 add mul 2 div}{P}
```

declares the node P at the point x=0.5 on the graph. It has the same effect as

```
\pnode(!/x 0.5 def x x x 1 add mul 2 div}){P}
```

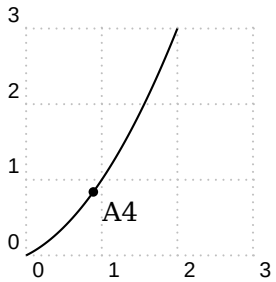
If your expression in `t` is algebraic, you must specify the keyword `algebraic`, as in `\fnpnode[algebraic]{0.5}{x*(x+1)/2}{P}`.




```
\begin{pspicture}[showgrid=true](-.5,-.5)(2.5,3)
\def\exn{x x 1 add mul 2 div}
\psplot{0}{2}{\exn}
\fnpnode{0.5}{\exn}{Q}
\psdot(Q)\uput[-45](Q){Q}
\end{pspicture}
```

`\fnpnodes{`*xmin*`}{`*xmax*`}{`*expression in x*`}{`*nodeRoot*`}`

Is similar to `\curvenodes`, but for the graph of a function. The keyword `algebraic` must be specified if your expression is indeed algebraic.




```
\begin{pspicture}[showgrid=true](-.5,-.5)(2.5,3)
\def\exn{x x 1 add mul 2 div}
\psplot{0}{2}{\exn}
\fnpnodes[plotpoints=10]{0}{2}{\exn}{A}
\psdot(A4)\uput[-45](A4){A4}
\end{pspicture}
```

`\shownode(`*P*`)`

is a debugging tool, which displays in the console window the coordinates of node P. This will not appear until the final stage of processing the PostScript file. You will get a PostScript error if the node you specify is undefined.

`\getnodelist{`*node root name*`}{`*next command*`}`

is useful in writing pstricks macros, where there is a list of parenthesized coordinates to be read and turned into a node sequence, following which `<next command>` is followed.

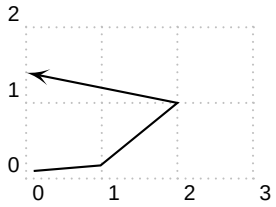
`\pnodes{`*P*`}(`*1,2*`)(`*2;3*`)...`

is effectively `\getnodelist{`*P*`}{}(1,2)(2;3)...`, just a quick way to turn a list of coordinates into a node sequence P0 P1 ...

`\psnline [Options] {`*arrows*`}(`*coors*`){`*name*`}`

for example, expects that there are nodes named P3..P8, and gives the same result as

```
\psline[linewidth=1pt]{->}(P3)(P4)(P5)(P6)(P7)(P8)
```




```
\begin{pspicture}[showgrid=true](-.5,-.5)(2.5,2)
\pnodes{P}(.1,.1)(1;10)(*2 {x^2/4})(0,1.4)
%defines P0..P3--now join them
\psnline[arrowscale=2]{-D>}(0,3){P}
\end{pspicture}
```

`\psLCNodeVar(`*node A*`)(`*node B*`)(`*factorA,factorB*`){`*node name*`}`

is similar to `\psLCNode`, and provides a means of forming a linear combination of two nodes, thought of as vectors. Where

```
\psLCNode(A){a}(B){b}{C}
```

effectively makes `C=aA+bB`,

```
\psLCNodeVar(A)(B)(a,b){C}
```

does the same, but the third argument `(a,b)` may be specified in any form acceptable to `\SpecialCoor`. (With `\psLCNode`, each coefficient may be specified in PostScript code.) One other difference is that `\psLCNodeVar` allows the reuse of a node name in place. For example, it is possible to write

```
\psLCNodeVar(A)(B)(2,3){A}% symbol A reassigned
```

where the equivalent in `\psLCNode` will lead to a PostScript error. Since `\AtoB` and `\AplusB` are defined using `\psLCNodeVar`, they also allow node name reuse: `\AtoB(Q)(P){P}` is legal.

`\psRelNodeVar(`*node A*`)(`*node B*`)(`*radius;angle*`){`*node name*`}`

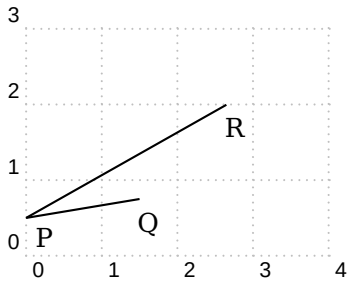
is similar to `\psRelNode`, and provides a means of scaling and rotating a line segment AB about A. The effect of

```
\psRelNodeVar(A)(B)(2;30){C}
```

is the same as

```
\psRelNode[angle=30](A)(B){2}{C}
```

but the third argument (2;30) may be specified in any form acceptable to \SpecialCoor, while specifying the angle argument in \psRelNode using PostScript is not possible. Note that \psRelNodeVar(0,0)(A)(B){C} may be interpreted as defining C to be the complex product of A and B.




```
\begin{pspicture}[showgrid=true](-.5,-.5)(3.5,3)
\pnode(0,.5){P}\pnode(1.5,.75){Q}
\psRelNodeVar(P)(Q)(2;20){R}
\psline(Q)(P)\uput[-45](P){P}
\uput[-70](Q){Q}
\psline(P)(R)\uput[-70](R){R}
\end{pspicture}
```

\psRelLineVar(*node A*)(*node B*)(*radius;angle*){*node name*}

stands to \psRelLine as \psRelNodeVar stands relative to \psRelNode.

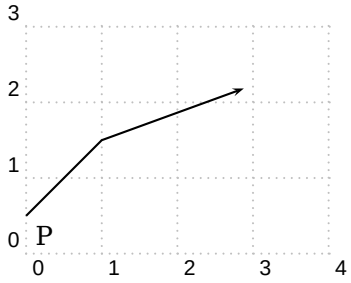
\psRelLineVar(A)(B)(a;b){C} defines the node C, and, in addition, draws the line segment AC.

\rhombus{*edge length*}(*A*){*B*}{*C*}{*D*}

computes the two remaining vertices C, D given two opposing vertices A, B of a rhombus with specified edge length. It does not draw the rhombus, which could be handled easily by \psline. Internally, \rhombus uses \psRelNodeVar.

\psrline(*P*)(*Q*)...

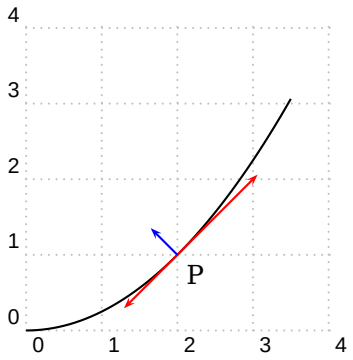
is like \psline, but drawing a line starting at (P), with successive increments (Q)... It has the same options as\psline.




```
\begin{pspicture}[showgrid=true](-.5,-.5)(3.5,3)
\pnode(0,.5){P}\pnode(1,1){Q}
\psrline{->}(P)(Q)(2;20)
\uput[-45](P){P}
\end{pspicture}
```

\psxline(*basept*){*nodexpr1*}{*nodexpr2*}

The x here stands for expression. The idea is that one builds a line from <basept>+<nodexpr1> to <basept>+<nodexpr2>.




```
\begin{pspicture}[showgrid=true](-.5,-.5)(3.5,4)
\def\pfn{t | t^2/4}
\psparametricplot[algebraic]{0}{3.5}{\pfn}
\curvepnode{2}{\pfn}{P}% sets P, Ptang
\normalvec(Ptang){Q}\uput[-45](P){P}
\psxline[linecolor=red]{<->}(P){-(Ptang)}{1.5(
    Ptang)}
\psxline[linecolor=blue]{->}(P){}{.5(Q)}%can use
    {} for {(0,0)}
\end{pspicture}
```

\polyIntersections{*Name1*}{*Name2*}(*A*)(*B*){*P*}...
\polyIntersections{*Name1*}{*Name2*}(*A*)(*B*){*P*}{*n*}

is the most complicated macro in the collection. It has two forms.

`\polyIntersections{<Name1>}{<Name2>}(A)(B)(1,2)(3;30)(6,5)...`

defines the polyline `L=(1,2)(3;30)(6,5)...`, and computes the two points of intersection closest to `A` in each direction with the directed line starting at A heading toward B. The first intersection point in the positive direction is named `<Name1>`, and the first intersection point in the opposite direction (from A) is named `<Name2>`. If one or other of these intersections is empty, the nodes are set to remote points on the line `AB`. The effect of the line joining the constructed nodes depends on the location of `A` and `B` relative to `L`, with two cases worth noting.

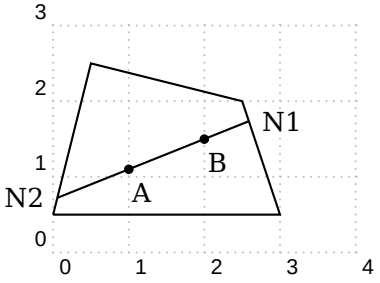
- if `L` is closed and if `A, B` are interior to one of its components, the resulting line extends across that component of `L`, and contains `AB`.
- If `L` is simple and closed, one of `A, B` is inside and the other outside, the resulting line segment will contain `A` but not `B`.

`\polyIntersections{<Name1>}{<Name2>}(A)(B){P}{n}`

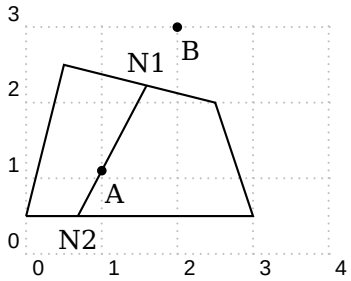
has exactly the same effect as

`\polyIntersections{<Name1>}{<Name2>}(A)(B)(P0)(P1)...(Pn)`

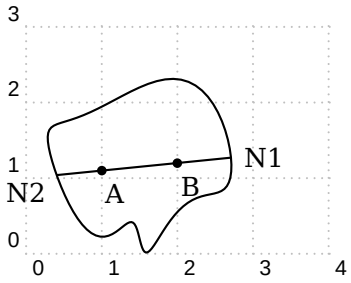
assuming `P0...Pn` to be previously defined nodes.



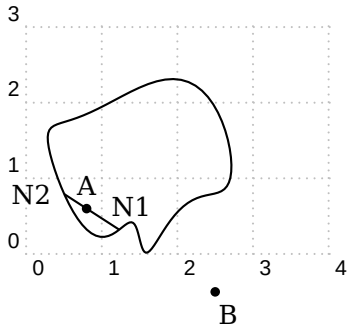

```
\begin{pspicture}[showgrid=true](-.5,-.5)
    (3.5,3)
\pnodes{P}(0,.5)(3,.5)(2.5,2)(.5,2.5)(0,.5)
\pnode(1,1.1){A}\pnode(2,1.5){B}
\polyIntersections{N1}{N2}(A)(B){P}{4}
\psnline(0,4){P}
\psdots(A)(B)\psline(N1)(N2)
\uput[-60](A){A}\uput[-60](B){B}
\uput[0](N1){N1}\uput[-180](N2){N2}
\end{pspicture}
```



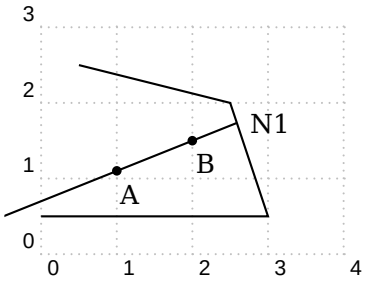

```
\begin{pspicture}[showgrid=true](-.5,-.5)
    (3.5,3)
\pnodes{P}(0,.5)(3,.5)(2.5,2)(.5,2.5)(0,.5)
\pnode(1,1.1){A}\pnode(2,3){B}
\polyIntersections{N1}{N2}(A)(B){P}{4}
\psnline(0,4){P}
\psdots(A)(B)\psline(N1)(N2)
\uput[-60](A){A}\uput[-60](B){B}
\uput[90](N1){N1}\uput[-90](N2){N2}
\end{pspicture}
```



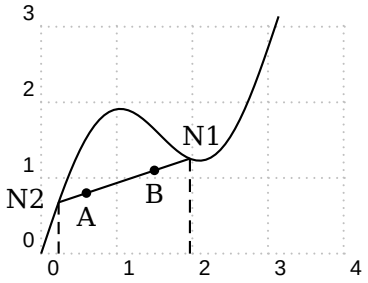

```
\begin{pspicture}[showgrid=true](-.5,-.5)
    (3.5,3)
\def\fn{1.5+sin(t)+.4*sin(2*t)%
 | 1+cos(t)+.2*cos(2*t)+.2*sin(4*t)}%
\pnode(1,1.1){A}\pnode(2,1.2){B}
\psset{plotpoints=100}
\psparametricplot[algebraic]{0}{6.283}{\fn}
\curvepnodes{0}{6.283}{\fn}{Z}
\polyIntersections{N1}{N2}(A)(B){Z}{99}
\psdots(A)(B)\psline(N1)(N2)
\uput[-60](A){A}\uput[-60](B){B}
\uput[0](N1){N1}\uput[220](N2){N2}
\end{pspicture}
```




```
\begin{pspicture}[showgrid=true](-.5,-.5)
    (3.5,3)
\def\fn{1.5+sin(t)+.4*sin(2*t)%
 | 1+cos(t)+.2*cos(2*t)+.2*sin(4*t)}%
\pnode(.8,.6){A}\pnode(2.5,-.5){B}
\psset{plotpoints=100}
\psparametricplot[algebraic]{0}{6.283}{\fn}
\curvepnodes{0}{6.283}{\fn}{Z}
\polyIntersections{N1}{N2}(A)(B){Z}{99}
\psdots(A)(B)\psline(N1)(N2)
\uput[90](A){A}\uput[-60](B){B}
\uput[70](N1){N1}\uput[180](N2){N2}
\end{pspicture}
```




```
\begin{pspicture}[showgrid=true](-.5,-.5)
    (3.5,3)
\pnodes{P}(0,.5)(3,.5)(2.5,2)(.5,2.5)(0,.5)
\pnode(1,1.1){A}\pnode(2,1.5){B}
\polyIntersections{N1}{N2}(A)(B){P}{3}
\psnline(0,3){P}
\psdots(A)(B)
\psclip{\psframe[linestyle=none](-.5,-.5)
    (3.5,2.5)}
\psline(N1)(N2)\endpsclip
\uput[-60](A){A}\uput[-60](B){B}
\uput[0](N1){N1}\uput[-180](N2){N2}
\end{pspicture}
```



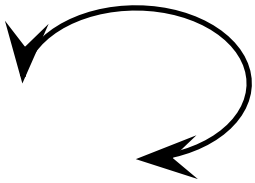

```
\begin{pspicture}[showgrid=true](-.5,-.5)
    (3.5,3)
\def\fn{x+sin(2*x)}
\psplot[algebraic]{0}{3.14}{\fn}
\fnpnodes[algebraic]{0}{3.14}{\fn}{P}
\pnode(.6,.8){A}\pnode(1.5,1.1){B}
\polyIntersections{N1}{N2}(A)(B){P}{49}
\psdots(A)(B)
\uput[-90](A){A}\uput[-90](B){B}
\psline(N1)(N2)
\psset{linestyle=dashed}
\psline(N1)(N1 | 0,0)
\psline(N2)(N2 | 0,0)
\uput[70](N1){N1}\uput[170](N2){N2}
\end{pspicture}
```

`\ArrowNotch{<NodeName>}{<nodeindex>}{<direction>}{<Notch>}`

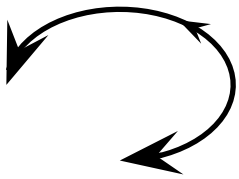
takes as inputs the root name of the node sequence, the index at which the arrow tip is to be drawn, and the direction (one of `>`,`<`) of the arrow. It then constructs the notch as a node with name `<Notch>`. The arrowhead may then be drawn with a command like `\psline{->}(N)(P3)`, assuming the tip was to be `P3` and the notch was `N`. Keep in mind that the macro takes its settings for linewidth, arrowscale, etc from the current values, so it is generally preferable to include them in a `\psset` before drawing the curve and calling `\ArrowNotch`. (Alternatively, they may be included as optional settings in `\ArrowNotch`.) The first example below shows a case where the native arrow direction is not good. The second shows how to make a version using `\ArrowNotch`. Notice that the minimum and maximum parameter values in the second example had to be modified to keep the curve from protruding near the end arrowheads.

`\ArrowNotch` is a computationally expensive macro (quadratic in `plotpoints`) designed to improve the placement of arrows on curves in those cases (high curvature, large values of linewidth, arrowscale, etc) where the native arrow direction is not optimal. The macro depends on the construction of a node sequence, say `P0..Pn`, of samples

of the curve (eg, with `\curvepnodes`) from which it computes the position of the notch of the arrow so that, when drawn, the arrow notch will be located on the curve in all cases. It operates with only two particular arrow shapes—those arrows specified with either `->` or `-D>`, or their reverses.



```
\def\fn{1.5+1.5*cos(t) | 1+sin(t)}
\psset{linewidth=2pt,arrowscale=3}
\begin{pspicture}(0,0)(3.5,3)
\psparametricplot[algebraic,arrows=<->]{PiDiv2
    neg}{Pi}{\fn}
\end{pspicture}
```



```
\def\fn{1.5+1.5*cos(t) | 1+sin(t)}
\psset{linewidth=2pt,arrowscale=3}
\begin{pspicture}(0,0)(3.5,3)
\curvepnodes{PiDiv2 neg}{Pi}\fn{P}%create P0..
   P49
\ArrowNotch{P}{0}{<}{Q}
\ArrowNotch{P}{49}{>}{R}
\ArrowNotch[arrowscale=1.5]{P}{27}{>}{S}
\psparametricplot[algebraic]{-1.47}{2.95}{\fn}
\psline{->}(Q)(P0)
\psline{->}(R)(P49)
\psline[arrowscale=1.5]{->}(S)(P27)
\end{pspicture}
```

10. List of all optional arguments for pst-node

Key	Type	Default
trueAngle	boolean	true
nodealign	boolean	[none]
href	ordinary	0
vref	ordinary	0.7ex
framesize	ordinary	[none]
nodesepA	ordinary	0pt
nodesepB	ordinary	0pt
nodesep	ordinary	0pt
armA	ordinary	10pt
armB	ordinary	10pt
arm	ordinary	10pt
XarmA	ordinary	
XarmB	ordinary	
Xarm	ordinary	[none]
YarmA	ordinary	
YarmB	ordinary	
Yarm	ordinary	
offsetA	ordinary	0pt
offsetB	ordinary	0pt
offset	ordinary	0pt
angleA	ordinary	0
angleB	ordinary	0
angle	ordinary	0pt
arcangleA	ordinary	8
arcangleB	ordinary	8
arcangle	ordinary	8
ncurvA	ordinary	0.67
ncurvB	ordinary	0.67
ncurv	ordinary	0.67
pcRef	boolean	true
lineAngle	ordinary	0
loopsize	ordinary	[none]
boxheight	ordinary	0.4cm
boxdepth	ordinary	0.4cm
boxsize	ordinary	0.4cm
nrot	ordinary	0
npos	ordinary	
tpos	ordinary	0.5
shortput	ordinary	none
colsep	ordinary	1.5cm
rowsep	ordinary	1.5cm

Continued on next page

Continued from previous page

Key	Type	Default
`name`	`ordinary`	`\relax`
`mcol`	`ordinary`	`c`
`mnodesize`	`ordinary`	`-1pt`
`mnode`	`ordinary`	`R`
`emnode`	`ordinary`	`none`

References

[1] Denis Girou. Présentation de PSTricks. *Cahier GUTenberg*, 16:21–70, April 1994.

[2] Michel Goosens, Frank Mittelbach, Sebastian Rahtz, Dennis Roegel, and Herbert Voß. *The LaTeX Graphics Companion*. Addison-Wesley Publishing Company, Boston, Mass., second edition, 2007.

[3] Nikolai G. Kollock. *PostScript richtig eingesetzt: vom Konzept zum praktischen Einsatz*. IWT, Vaterstetten, 1989.

[4] Herbert Voß. *`PSTricks` – Grafik für TeX und LaTeX*. DANTE – Lehmanns, Heidelberg/Hamburg, fifth edition, 2008.

[5] Timothy Van Zandt. *`multido.tex` - a loop macro, that supports fixed-point addition*. CTAN:/macros/generic/multido.tex, 1997.

[6] Timothy Van Zandt and Denis Girou. Inside PSTricks. *TUGboat*, 15:239–246, September 1994.

Index