PSTricks:

PostScript macros for Generic TeX

Documented Code

Timothy Van Zandt[1]

25 March 191997
Version 0.93a-97

PSTricks is a collection of PostScript macros that is compatible with most TEX macro packages, including Plain TEX and LATEX. Included are macros for color, graphics, rotation and overlays.

This is the documented code. There is also a *User's Guide* and a read-me file.

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

Contents

1 Disclaimers, Guidelines and Other Comments

Disclaimer These macros are extensive and were written hurriedly. Only modest attempts have been made to clean up and optimize the code. The internals may change substantially until version 1.0 comes up.

PostScript Guidelines The following guidelines were followed for macros using PostScript `\special`'s:

1. Almost no `gsave` and `grestore` commands are used (reducing the likelihood of conflicts with dvi-to-ps drivers or an unmatched `gsave` or `grestore` ending up on a page).
2. Most end-user macros (those without `@`) have error-checking so that bad arguments or other misuse will not generate PostScript errors.

Macros A "macro" means any command sequence that is documented in this with a heading entry. Macros with `@` are internal, and others are part of the user interface. Commands that are not in the heading preceding their definition are internal commands of a macro, and are not meant to be used directly by other macros.

Local and global variables There are various classes of scratch registers and commands:

Global These can be changed using `\global`, etc.

```
\pst@tempg
\pst@temph
\pst@dimg
\pst@dimh
\pst@cntg
\pst@cnth
\pst@boxg
```

Local-I These cannot be changed with `\global`, but otherwise there are no restrictions.

```
\next
\@tempa
```

Local-I Changes to these must be local to the macro in which they occur (be grouping).

```
\pst@tempa
\pst@tempb
\pst@tempc
\pst@tempd
```

Local-II Changes to these must be local to the macro in which they occur, and it must be possible to use these as arguments of macros.

```
\pst@dima
\pst@dimb
\pst@dimc
\pst@dimd
\if@pst
```

There is one exception. When using these in coordinates that are processed directly as Cartesian coordinates rather than with `\pst@getcoor`, they must be used in this order:

```
(\pst@dima,\pst@dimb)(\pst@dimc,\pst@dimd)
```

Shared These are used to share information between macros. Their value may be set by one macro and then used by another. Use with care. Do not set with `\global`.

command	*usage*
`\pst@hbox`	Box created and manipulated in HR-box macros.
`\pst@coor`	PostScript code for a coordinate.
`\pst@angle`	PostScript code for an angle.
`\pst@rot`	PostScript code for a rotation angle.
`\if@star`	This is a flag to keep track of optional `*`.

Plain TEX commands The commands

```
\newbox
\newcount
\newdimen
\newif
\loop ... \repeat ... \fi
\z@
\sixt@@n
```

are defined in `plain.tex` are part of most macro packages. PSTricks assumes that they are defined. Other than these, PSTricks only makes use of TEX primitives.

Dividing the file *Breaking up the file* `pstricks.tex` can be broken up into the following components:

Basics (Including color and simple rotation.) Sections 2, 7, 8, **??**, 11, **??**, 34 and 39.

Graphics Sections 12, 21, 17, 19, 26, 28, 18, **??**, 30 and 31. Requires also **Basics**.

Rotation (Including picture environment.) Sections **??**, **??**, **??** and 37. Requires also **Basics**.

2 Preliminaries

Check whether file has been loaded already.

```
\csname PSTricksLoaded\endcsname
\let\PSTricksLoaded\endinput
```

Take care of the catcode of `@`:

```
\edef\PstAtCode{\the\catcode`\@}
\catcode`\@=11\relax
```

Here are some hacks borrowed from LaTeX, which are defined if LaTeX is not being used.

```
\expandafter\ifx\csname @latexerr\endcsname\relax
  \long\def\@ifundefined#1#2#3{\expandafter\ifx\csname
    #1\endcsname\relax#2\else#3\fi}
  \def\@namedef#1{\expandafter\def\csname #1\endcsname}
  \def\@nameuse#1{\csname #1\endcsname}
  \def\@eha{%
    Your command was ignored.^^J
    Type \space I <command> <return> \space to replace
    it with another command,^^J
    or \space <return> \space to continue without it.}
  \def\@spaces{\space\space\space\space}
  \def\typeout#1{\immediate\write\@unused{#1}}
  \alloc@7\write\chardef\sixt@@n\@unused
  \def\@empty{}
  \def\@gobble#1{}
  \def\@nnil{\@nil}
  \def\@ifnextchar#1#2#3{%
    \let\@tempe#1\def\@tempa{#2}\def\@tempb{#3}\futurelet\@tempc\@ifnch}
  \def\@ifnch{%
    \ifx\@tempc\@sptoken
      \let\@tempd\@xifnch
    \else
      \ifx\@tempc\@tempe \let\@tempd\@tempa \else \let\@tempd\@tempb \fi
    \fi
    \@tempd}
  \begingroup
    \def\:{\global\let\@sptoken= } \:
    \def\:{\@xifnch} \expandafter\gdef\: {\futurelet\@tempc\@ifnch}
  \endgroup
\fi
```

Announce that the file is being loaded:

```
\typeout{‘PSTricks’ v\fileversion\space\space <\filedate> (tvz)}
```

3 Error messages

\@pstrickserr

`\@pstrickserr` is analogous to `\@latexerr`.

```
\def\@pstrickserr#1#2{%
  \begingroup
    \newlinechar‘\^^J
    \edef\pst@tempc{#2}%
    \expandafter\errhelp\expandafter{\pst@tempc}%
    \typeout{%
      PSTricks error. \space See User’s Guide for further information.^^J
      \@spaces\@spaces\@spaces\@spaces
      Type \space H <return> \space for immediate help.}%
    \errmessage{#1}%
  \endgroup}
```

\@ehpa, \@ehpb, \@ehpc

Here are some extra \errhelp message:

```
\def\@ehpa{%
  Your command was ignored. Default value substituted.^^J
  Type \space <return> \space to procede.}
\def\@ehpb{%
  Your command was ignored. Will recover best I can.^^J
  Type \space <return> \space to procede.}
\def\@ehpc{%
  You better fix this before proceding.^^J
  See the PSTricks User's Guide or ask your system administrator for help.^^J
  Type \space X <return> \space to quit.}
```

\pst@misplaced

```
\def\pst@misplaced#1{\@pstrickserr{Misplaced \string#1 command}\@ehpb}
```

4 Scratch registers

```
\newdimen\pst@dima
\newdimen\pst@dimb
\newdimen\pst@dimc
\newdimen\pst@dimd
\newdimen\pst@dimg
\newdimen\pst@dimh
\newbox\pst@hbox
\newbox\pst@boxg
\newcount\pst@cnta
\newcount\pst@cntb
\newcount\pst@cntc
\newcount\pst@cntd
\newcount\pst@cntg
\newcount\pst@cnth
\newif\if@pst
```

5 Useful hacks

\pst@ifstar, \if@star

```
\newif\if@star
\def\pst@ifstar#1{%
  \@ifnextchar*{\@startrue\def\next*{#1}\next}{\@starfalse#1}}
```

\pst@expandafter

```
\def\pst@expandafter#1#2{%
  \def\next{#1}%
  \edef\@tempa{#2}%
  \ifx\@tempa\@empty
    \@pstrickserr{Unexpected empty argument!}\@ehpb
    \def\@tempa{\@empty}%
  \fi
  \expandafter\next\@tempa}
```

6 Arithmetic

\pst@dimtonum, \pst@@dimtonum

This macro strips the value of #1, a dimension register, of the pt, and assigns the result to #2, a command sequence. This is used for arithmetic and for converting TEX dimensions to PostScript.

```
\def\pst@dimtonum#1#2{\edef#2{\pst@@dimtonum#1}}
\def\pst@@dimtonum#1{\expandafter\pst@@@dimtonum\the#1}
{\catcode`\p=12 \catcode`\t=12 \global\@namedef{pst@@@dimtonum}#1pt{#1}}
```

\pst@pyth

This is a piecewise-linear approximation to $(\#1^2 + \#2^2)^(1/2)$. The answer is assigned to #3. All arguments should be dimension registers.

```
\def\pst@pyth#1#2#3{%
  \ifdim#1>#2\pst@@pyth#1#2#3\else\pst@@pyth#2#1#3\fi}
\def\pst@@pyth#1#2#3{%
  \ifdim4#1>9#2%
    #3=#1\advance#3 .2122#2%
  \else
    #3=.8384#1\advance#3 .5758#2%
  \fi}
```

\pst@divide

This computes #3=#1/#2 reasonably quickly. #1 and #2 should be dimensions, and #3 should be a command sequence.

```
\def\pst@divide#1#2#3{%
  \begingroup
    \pst@dimg=#1\relax\pst@dimh=#2\relax
    \pst@cnta=\pst@dimg
    \pst@cntb=1073741824
    \pst@cntc=65536
    \def\pst@tempa{\fi\ifnum}%
    \loop\ifnum\pst@cnta<\pst@cntb
      \pst@tempa\pst@cntc>\@ne
      \multiply\pst@cnta2\divide\pst@cntc2
    \repeat
    \divide\pst@dimh\pst@cntc
    \divide\pst@cnta\pst@dimh
    \global\pst@dimg\number\pst@cnta sp
  \endgroup
  \pst@dimtonum\pst@dimg#3}
```

7 Configuration file

\pst@configerr

```
\def\pst@configerr#1{%
  \@pstrickserr{\string#1 not defined in pstricks.con}\@ehpc}
```

```
\def\pstVerb#1{\pst@configerr\pstVerb}
\def\pstverb#1{\pst@configerr\pstverb}
\def\pstverbscale{\pst@configerr\pstverbscale}
\def\pstrotate{\pst@configerr\pstrotate}
\def\pstheader#1{\pst@configerr\pstheader}
\def\pstdriver{\pst@configerr\pstdriver}
\@ifundefined{pstcustomize}%
  {\def\pstcustomize{\endinput\let\pstcustomize\relax}}{}
\input pstricks.con
```

\PSTricksOff

```
\newif\ifPSTricks
\PSTrickstrue
\def\PSTricksOff{%
  \def\pstheader##1{}%
  \def\pstverb##1{}%
  \def\pstVerb##1{}%
  \PSTricksfalse}
```

8 PostScript header file

\pst@def, \pst@ATH, \pst@useheader

There are three ways to use PSTricks:

1. Use `pstricks.doc` directly. No header file is used.
2. Use `\pst-make.tex` to generate a stripped input file for use without a header.
3. Use `\pst-make.tex` to generate a header and a stripped input file for use with the header.

PSTricks has been optimized for use with a header file (and the difference is speed and memory is very significant), but the flexible system described above makes it easier to maintain the code and allows one to use PSTricks with a DVI-to-PS driver that does not support header files.

The following commands should be used in this `.doc` file for PostScript macros and other goodies related to the header file. Their behavior for each of the three cases list above is given below. These commands should always come *at the beginning of the line*, and should not inside conditionals.

- `\pst@def{foo}<bar>`
  1. `\tx@foo` is defined to be `bar`.
  2. Writes
     ```
     \def\tx@foo{bar}
     ```
     to `pstricks.tex`.
  3. Writes
     ```
     /foo { bar } def
     ```
     to `pstricks.pro` and
     ```
     \def\tx@foo{foo}
     ```

to `pstricks.tex`.

- `\pst@ATH<foo>`

  1. Gobbles `foo`.
  2. Ignores line.
  3. Writes `foo` to `pstricks.pro`.

  Note: `\pst@ATH` must come at the beginning of the line.

- `\ifx\pst@useheader\iftrue foo \else bar \fi`

  1. Ignores `foo` and includes `bar`.
  2. Ignores `foo` and processes `bar`.
  3. Processes `foo` and ignores `bar`.

  Note: `\ifx\pst@useheader\iftrue`, `\else` and `\fi` must each be on their own line.

`pst@make.tex` can be used to process other files at well, in the right order.

```
\@ifundefined{pst@def}{\def\pst@def#1<#2>{\@namedef{tx@#1}{#2 }}}{}
\@ifundefined{pst@ATH}{\def\pst@ATH<#1>{}}{}
```

\pst@dict

The PostScript dictionary `tx@Dict` is set up in the header file, if one is used. Otherwise, it is set up with each procedure that uses the dictionary, if it is not currently defined.

```
\pst@ATH<\% Version \fileversion, \filedate.>
\pst@ATH<\% For use with \pstdriver.>
\pst@ATH</tx@Dict 200 dict def tx@Dict begin>
\pst@ATH</ADict 25 dict def>
\pst@ATH</CM { matrix currentmatrix } bind def>
\pst@ATH</SLW /setlinewidth load def>
\pst@ATH</CLW /currentlinewidth load def>
\pst@ATH</CP /currentpoint load def>
\pst@ATH</ED { exch def } bind def>
\pst@ATH</L /lineto load def>
\pst@ATH</T /translate load def>
\ifx\pst@useheader\iftrue
  \pstheader{pstricks.pro}
  \def\pst@dict{tx@Dict begin }
  \def\pst@theheaders{pstricks.pro}
\else
  \def\pst@dict{%
    /tx@Dict where
    { pop tx@Dict begin}
    { userdict begin
        /tx@Dict 200 dict def
      end
      tx@Dict begin
        /ADict 25 dict def
        /CM { matrix currentmatrix } bind def
```

```
        /SLW /setlinewidth load def
        /CLW /currentlinewidth load def
        /CP /currentpoint load def
        /ED { exch def } bind def
        /L /lineto load def }
    ifelse }
  \def\pst@theheaders{}%
\fi
```

\pst@Verb

```
\def\pst@Verb#1{\pstVerb{\pst@dict #1 end}}
```

9 PostScript hacks

\tx@Atan, \tx@Div

These are variants of `atan`, and `div`, that recover when result is not defined.

```
\pst@def{Atan}</atan load stopped { pop pop 0 } if>
\pst@def{Div}<dup 0 eq { pop } { div } ifelse>
```

\tx@NET

```
\pst@def{NET}<neg exch neg exch T>
```

\tx@Pyth

A B `Pyth` $(A^2 + B^2)^{1/2}$

```
\pst@def{Pyth}<dup mul exch dup mul add sqrt>
```

\tx@PtoC

Polar to Cartesian:

r a `PtoC` x y

```
\pst@def{PtoC}<2 copy cos mul 3 1 roll sin mul>
```

\tx@PathLength

`PathLength` is taken from the Blue Book. It leaves on the stack the length of the current path.

```
\pst@def{PathLength@}<%
  /z z y y1 sub x x1 sub \tx@Pyth add def
  /y1 y def /x1 x def>
\pst@def{PathLength}<%
  flattenpath /z 0 def
  { /y1 ED /x1 ED /y2 y1 def /x2 x1 def } % moveto
  { /y ED /x ED \tx@PathLength@ } % lineto
  {} % curveto; ignore because of flattenpath.
  { /y y2 def /x x2 def \tx@PathLength@ } % closepath
  pathforall z>
```

10 Converting TEX things to PostScript

\pst@number, \tx@STP, \tx@STV

- PSTricks' PostScript unit is 1pt, rather than 1bp, because this is more efficient.
- \pst@number{<dimen register>} converts *dimen* to PostScript, in points (pt).
- \tx@STP scales the DVI-to-PS driver's \pstverb environment to points (pt).
- \tx@STV scales the DVI-to-PS drivers's ungrouped PostScript \special environment (\pstVerb) to points (pt).

```
\pst@dimg=\pstunit\relax
\ifdim\pst@dimg=1bp
  \def\pst@stp{.996264 dup scale}
\else
  \edef\pst@stp{1 \pst@@dimtonum\pst@dimg\space div dup scale}
\fi
\pst@def{STP}<\pst@stp>
\pst@def{STV}<\pstverbscale\space\tx@STP>
\def\pst@number#1{\pst@@dimtonum#1\space}
```

\pst@checknum

The first argument of \pst@checknum should be a number, and the second argument is a command. There are three possible outcomes:

1. The number is suitable for PostScript consumption, the command is set to the number, and \pst@num is set to 1 if the number is positive and to 2 if the number is negative.
2. \pst@checknum detects that the number is not suitable for PostScript; \pst@num is set to 0, an error is given, and the command is defined to be 0 .
3. The number is not suitable for PostScript consumption, but \pst@checknum does not detect this. \pst@num is set to 1 or 2, and the command is set to some number that *is* suitable for PostScript.

A trailing space is always added.

\pst@checknum should generate no extraneous errors nor output, even if the first argument is a bad number.

This macro is probably pretty close to optimal for what it does, as many variations have been tried.

```
\def\pst@checknum#1#2{%
  \edef\next{#1}%
  \ifx\next\@empty
    \let\pst@num\z@
  \else
    \expandafter\pst@@checknum\next..\@nil
  \fi
  \ifnum\pst@num=\z@
    \@pstrickserr{Bad number: `#1'. 0 substituted.}\@ehpa
```

```
    \def#2{0 }%
  \else
    \edef#2{\ifnum\pst@num=2 -\fi\the\pst@cntg.%
      \expandafter\@gobble\the\pst@cnth\space}%
  \fi}
\def\pst@@checknum{%
  \@ifnextchar-%
    {\def\pst@num{2}\expandafter\pst@@@checknum\@gobble}%
    {\def\pst@num{1}\pst@@@checknum}}
\def\pst@@@checknum#1.#2.#3\@nil{%
  \afterassignment\pst@@@@checknum\pst@cntg=0#1\relax\@nil
  \afterassignment\pst@@@@checknum\pst@cnth=1#2\relax\@nil}
\def\pst@@@@checknum#1\relax\@nil{%
  \ifx\@nil#1\@nil\else\let\pst@num\z@\fi}
```

\pst@getnumii, \pst@getnumiii, \pst@getnumiv

These are for processing comma-separated lists of numbers. They assign the numbers to \pst@tempg, \pst@temph, etc. Use like

```
\pst@expandafter\pst@getnumiii{foo} {} {} {} {}\@ni
```

If there are too few numbers, an error results. If there are too many, the extra numbers are ignored.

```
\def\pst@getnumii#1 #2 #3\@nil{%
  \pst@checknum{#1}\pst@tempg
  \pst@checknum{#2}\pst@temph}
\def\pst@getnumiii#1 #2 #3 #4\@nil{%
  \pst@checknum{#1}\pst@tempg
  \pst@checknum{#2}\pst@temph
  \pst@checknum{#3}\pst@tempi}
\def\pst@getnumiv#1 #2 #3 #4 #5\@nil{%
  \pst@checknum{#1}\pst@tempg
  \pst@checknum{#2}\pst@temph
  \pst@checknum{#3}\pst@tempi
  \pst@checknum{#4}\pst@tempj}
```

\pst@getdimnum

Like \pst@getnumii, but first item is a dimension and second is a number.

```
\def\pst@getdimnum#1 #2 #3\@nil{%
  \pssetlength\pst@dimg{#1}%
  \pst@checknum{#2}\pst@tempg}
```

\pst@getscale

\pst@getscale can have one or two numbers in its first argument.

```
\def\pst@getscale#1#2{%
  \pst@expandafter\pst@getnumii{#1 #1} {} {} {}\@nil
  \edef#2{\pst@tempg\space \pst@temph\space scale }%
  \ifdim\pst@tempg\p@=\z@
```

```
    \@pstrickserr{Bad scaling argument `#1'}\@ehpa
    \def#2{}%
  \else
    \ifdim\pst@temph\p@=\z@
      \@pstrickserr{Bad scaling argument}\@ehpa
      \def#2{}%
    \else
      \ifdim\pst@tempg\p@=\p@ \ifdim\pst@temph\p@=\p@ \def#2{}\fi\fi
    \fi
  \fi}
```

\pst@getint

```
\def\pst@getint#1#2{%
  \pst@cntg=#1\relax
  \edef#2{\the\pst@cntg\space}}
```

\pslbrace, \psrbrace

When balanced braces are used, they work without problems in `\special`'s. `\pslbrace` and `\psrbrace` let you use unbalanced braces.

```
\begingroup
  \catcode`\{=12
  \catcode`\}=12
  \catcode`\[=1
  \catcode`\]=2
  \gdef\pslbrace[{ ]
  \gdef\psrbrace[} ]
\endgroup
```

11 Colors

\@newcolor

`\@newcolor{<color>}{<spec>}`, where *color* is a name and *spec* is the associated PostScript color specification , sets

- `\<color>` to `\pst@color{<spec>}`, and
- `\color@<color>` to *spec*.

Then `\<color>` can be used by the user to color text, etc., and `\color@<color>` is used by PSTricks graphics objects to find the specification for *color*.

```
\def\@newcolor#1#2{%
  \expandafter\edef\csname #1\endcsname{\noexpand\pst@color{#2}}%
  \expandafter\edef\csname color@#1\endcsname{#2}%
  \ignorespaces}
```

\pst@color, \pst@endcolor

The argument of `\pst@color` should be a PostScript command for setting the color; e.g., `0 setgray`. It saves the command in `\pst@currentcolor`, and then switches to

\pst@currentcolor at the end of the current group. The color changes do not extend across pages, although this capability could be written into the output routines (so that \pst@currentcolor is set at the beginning of the page, and headers and footers begin with \black, etc.). Moving boxes cause problems, but there is no way around this until TEX supports color internally.

```
\def\pst@color#1{%
  \def\pst@currentcolor{#1}\pstVerb{#1}\aftergroup\pst@endcolor}
\def\pst@endcolor{\pstVerb{\pst@currentcolor}}
\def\pst@currentcolor{0 setgray}
```

\altcolormode, \pst@grestore

The color macros defined above can conflict with other color macros. \altcolormode sets up a different scheme that uses gsave and grestore to reset colors. This may reduce the likelihood of such conflict. It also makes moving boxes less of a problem, as long as the color command is itself grouped within the box. However, if the scope of a color command extends across pages in a TEX input file, unmatched gsave's and grestore's will be left on pages, wreaking havok on the output. \pst@grestore is defined to do various things that makes using grestore more robust.

```
\def\altcolormode{%
  \def\pst@color##1{%
    \pstVerb{gsave ##1}\aftergroup\pst@endcolor}%
  \def\pst@endcolor{\pstVerb{\pst@grestore}}}
\def\pst@grestore{%
  currentpoint
  matrix currentmatrix
  currentfont
  grestore
  setfont
  setmatrix
  moveto}
```

\pst@usecolor

This looks up the color specification.

```
\def\pst@usecolor#1{\csname color@#1\endcsname\space}
```

\newgray

\newgray uses PostScript's setgray operator.

```
\def\newgray#1#2{%
  \pst@checknum{#2}\pst@tempg
  \@newcolor{#1}{\pst@tempg setgray}}
```

\newrgbcolor

This works like \newgray, but the color specification should consist of 3 numbers rather than just 1, and the setrgbcolor operator is used.

```
\def\newrgbcolor#1#2{%
  \pst@expandafter\pst@getnumiii{#2} {} {} {} {}\@nil
  \@newcolor{#1}{\pst@tempg \pst@temph \pst@tempi setrgbcolor}}
```

\newhsbcolor

This is just like \newrgbcolor, but the sethsbcolor operator is used.

```
\def\newhsbcolor#1#2{%
  \pst@expandafter\pst@getnumiii{#2} {} {} {} {}\@nil
  \@newcolor{#1}{\pst@tempg \pst@temph \pst@tempi sethsbcolor}}
```

\newcmykcolor

This is like \newrgbcolor, the color specification consists of 4 numbers and the setcmykcolor operator is used.

```
\def\newcmykcolor#1#2{%
  \pst@expandafter\pst@getnumiv{#2} {} {} {} {} {}\@nil
  \@newcolor{#1}{\pst@tempg \pst@temph \pst@tempi \pst@tempj setcmykcolor}}
```

\black, \darkgray, \gray, \lightgray, \white

Here are some default gray definitions:

```
\newgray{black}{0}
\newgray{darkgray}{.25}
\newgray{gray}{.5}
\newgray{lightgray}{.75}
\newgray{white}{1}
```

\red, \green, \blue, \yellow, \cyan, \magenta

And some default rgb color definitions.

```
\newrgbcolor{red}{1 0 0}
\newrgbcolor{green}{0 1 0}
\newrgbcolor{blue}{0 0 1}
\newrgbcolor{yellow}{1 1 0}
\newrgbcolor{cyan}{0 1 1}
\newrgbcolor{magenta}{1 0 1}
```

12 Setting graphics parameters

\psset

For each *parameter*=*value* pair in its argument, \psset invokes

\psset@*parameter*{*value*}

The value is processed and typically stored in \ps<parameters> if the value is user-accessible and \psk@<parameter> if not. \psset ignores spaces that follow the comma that separates key-value pairs.

When initializing *parameter* in this file, preferable use

\psset@*parameter*{*value*}

so that default values can be easily extracted for the *User's Guide*.

```
\def\psset#1{\@psset#1,\@nil\ignorespaces}
\def\@psset#1,{%
  \@@psset#1==\@nil
  \@ifnextchar\@nil{\@gobble}{\@psset}}
\def\@@psset#1=#2=#3\@nil{%
  \@ifundefined{psset@#1}%
    {\@pstrickserr{Graphics parameter `#1' not defined.}\@ehpa}%
    {\@nameuse{psset@#1}{#2}}}%
```

\newpsstyle

```
\def\psset@style#1{%
  \@ifundefined{pscs@#1}%
    {\@pstrickserr{Custom style `#1' undefined}\@ehpa}%
    {\@nameuse{pscs@#1}}}
\def\newpsstyle#1#2{\@namedef{pscs@#1}{\psset{#2}}}
```

\@none

Use to check when a parameter value is `none`.

```
\def\@none{none}
```

\pst@getcolor

This is used by various graphics parameters that are colors.

```
\def\pst@getcolor#1#2{%
  \@ifundefined{color@#1}%
    {\@pstrickserr{Color `#1' not defined}\@eha}%
    {\edef#2{#1}}}
```

13 Dimensions

\psunit, \psxunit, \psyunit

```
\newdimen\psunit \psunit 1cm
\newdimen\psxunit \psxunit 1cm
\newdimen\psyunit \psyunit 1cm
\let\psrunit\psunit
```

\pssetlength, \psaddtolength, \pssetxlength, \pssetylength

```
\def\pstunit@off{\let\@psunit\ignorespaces\ignorespaces}
\def\pssetlength#1#2{%
  \let\@psunit\psunit
  \afterassignment\pstunit@off
  #1 #2\@psunit}
\def\psaddtolength#1#2{%
  \let\@psunit\psunit
  \afterassignment\pstunit@off
  \advance#1 #2\@psunit}
\def\pssetxlength#1#2{%
```

```
  \let\@psunit\psxunit
  \afterassignment\pstunit@off
  #1 #2\@psunit}
\def\pssetylength#1#2{%
  \let\@psunit\psyunit
  \afterassignment\pstunit@off
  #1 #2\@psunit}
```

\psset@unit, \psset@xunit, \psset@yunit

```
\def\psset@unit#1{%
  \pssetlength\psunit{#1}%
  \psxunit=\psunit
  \psyunit=\psunit}
\def\psset@runit#1{\pssetlength\psrunit{#1}}
\def\psset@xunit#1{\pssetxlength\psxunit{#1}}
\def\psset@yunit#1{\pssetylength\psyunit{#1}}
```

\pst@getlength, pst@@getlength

`#1` is a TEX dimensions. `\pst@getlength` sets `#2` to the PostScript code for `#1`, and `\pst@@getlength` set `#2` to the TEX code for `#1`.

```
\def\pst@getlength#1#2{%
  \pssetlength\pst@dimg{#1}%
  \edef#2{\pst@number\pst@dimg}}
\def\pst@@getlength#1#2{%
  \pssetlength\pst@dimg{#1}%
  \edef#2{\number\pst@dimg sp}}
```

14 Normal Coordinates and angles

\pst@getcoor, \pst@coor

`\pst@@getcoor` should be defined to read a coordinate and convert it to PostScript, assigning the result to `\pst@coor` (including the trailing space).

`\pst@getcoor` invokes `\pst@@getcoor` and then sets its second argument to `\pst@coor`.

```
\def\pst@getcoor#1#2{\pst@@getcoor{#1}\let#2\pst@coor}
\def\pst@coor{0 0 }
```

\pst@getcoors, \pst@coors

`\pst@getcoors` reads coordinates until there are none left, adding them *in reverse order* to `\pst@coors`.

```
\def\pst@getcoors#1#2{%
  \def\pst@aftercoors{\addto@pscode{#1 \pst@coors }#2}%
  \def\pst@coors{}%
  \pst@@getcoors}
\def\pst@@getcoors(#1){%
  \pst@@getcoor{#1}%
  \edef\pst@coors{\pst@coor\pst@coors}%
  \@ifnextchar({\pst@@getcoors}{\pst@aftercoors}}
```

\pst@getangle, \pst@angle

\pst@@getangle should be defined to read an angle and convert it to PostScript, assigning the result to \pst@angle (including the trailing space).

\pst@getangle invokes \pst@@getangle and then sets its second argument to \pst@angle.

```
\def\pst@getangle#1#2{\pst@@getangle{#1}\let#2\pst@angle}
\def\pst@angle{0 }
```

getcoor@c, \NormalCoor

By default, coordinates are read as Cartesian coordinates by \cartesian@coor.

Angles are read as numbers, scaled by \pst@angleunit.

\NormalCoor sets these two defaults, and also defines the translation for the put commands to be done by TEX using Cartesian coordinates.

```
\def\cartesian@coor#1,#2,#3\@nil{%
  \pssetxlength\pst@dimg{#1}%
  \pssetylength\pst@dimh{#2}%
  \edef\pst@coor{\pst@number\pst@dimg \pst@number\pst@dimh}}
\def\NormalCoor{%
  \def\pst@@getcoor##1{\pst@expandafter\cartesian@coor{##1},\relax,\@nil}%
  \def\pst@@getangle##1{%
    \pst@checknum{##1}\pst@angle
    \edef\pst@angle{\pst@angle \pst@angleunit}}%
  \def\psput@##1{\pst@@getcoor{##1}\leavevmode\psput@cartesian}}
\NormalCoor
```

\pst@angleunit, \degrees, \radians

\degrees sets \pst@angleunit to the PostScript code for scaling the angle, including the trailing space.

```
\def\degrees{\@ifnextchar[{\@degrees}{\def\pst@angleunit{}}}
\def\@degrees[#1]{%
  \pst@checknum{#1}\pst@tempg
  \edef\pst@angleunit{360 \pst@tempg div mul }%
  \ignorespaces}
\def\radians{\def\pst@angleunit{57.2956 mul }}
\def\pst@angleunit{}
```

15 Special coordinates and angles

This is a tedious but useful.

\SpecialCoor

```
\def\SpecialCoor{%
  \def\pst@@getcoor##1{\pst@expandafter\special@coor{##1}||\@nil}%
  \def\pst@@getangle##1{\pst@expandafter\special@angle{##1}\@empty)\@nil}%
  \def\psput@##1{\pst@@getcoor{##1}\leavevmode\psput@special}}
```

\specialcoor

```
\def\special@coor#1|#2|#3\@nil{%
  \ifx#3|\relax
    \mixed@coor{#1}{#2}%
  \else
    \special@@coor#1;;\@nil
  \fi}
\def\special@@coor#1{%
  \ifcat#1a\relax
    \def\next{\node@coor#1}%
  \else
    \ifx#1[\relax
      \def\next{\Node@coor[}%
    \else
      \ifx#1!\relax
        \def\next{\raw@coor}%
      \else
        \def\next{\special@@@coor#1}%
      \fi
    \fi
  \fi
  \next}
\def\special@@@coor#1;#2;#3\@nil{%
  \ifx#3;\relax
    \polar@coor{#1}{#2}%
  \else
    \cartesian@coor#1,\relax,\@nil
  \fi}
```

\mixed@coor

This allows mixing of coordinate types with `\SpecialCoor`.

```
\def\mixed@coor#1#2{%
    \begingroup
      \specialcoor@ii#1;;\@nil
      \let\pst@tempa\pst@coor
      \specialcoor@ii#2;;\@nil
      \xdef\pst@tempg{\pst@tempa pop \pst@coor exch pop }%
    \endgroup
    \let\pst@coor\pst@tempg}
```

\polar@coor

For polar coordinates

```
\def\polar@coor#1#2{%
  \pssetlength\pst@dimg{#1}%
  \pst@@getangle{#2}%
  \edef\pst@coor{\pst@number\pst@dimg \pst@angle \tx@PtoC}}
```

\raw@coor

For raw PostScript.

```
\def\raw@coor#1;#2\@nil{%
```

```
  \edef\pst@coor{%
    #1 \pst@number\psyunit mul exch \pst@number\psxunit mul exch }}
```

\node@coor, \Node@coor

These are defined in `pst-node.tex`.

```
\def\node@coor#1\@nil{%
  \@pstrickserr{You must load 'pst-node.tex' to use node coordinates.}\@ehps
  \def\pst@coor{0 0 }}
\def\Node@coor{\node@coor}
```

\special@angle

```
\def\special@angle#1#2)#3\@nil{%
  \ifx#1!\relax
    \edef\pst@angle{#2 \pst@angleunit}%
  \else
    \ifx#1(\relax
      \pst@@getcoor{#2}%
      \edef\pst@angle{\pst@coor exch \tx@Atan}%
    \else
      \pst@checknum{#1#2}\pst@angle
      \edef\pst@angle{\pst@angle \pst@angleunit}%
    \fi
  \fi}
```

\Cartesian, \Polar

These are obsolete.

```
\def\Cartesian{%
  \def\cartesian@coor##1,##2,##3\@nil{%
    \pssetxlength\pst@dimg{##1}%
    \pssetylength\pst@dimh{##2}%
    \edef\pst@coor{\pst@number\pst@dimg \pst@number\pst@dimh}}%
  \@ifnextchar({\Cartesian@}{}}
\def\Cartesian@(#1,#2){%
  \pssetxlength\psxunit{#1}%
  \pssetylength\psyunit{#2}%
  \ignorespaces}
\def\Polar{%
  \def\psput@cartesian{\psput@special}%
  \def\cartesian@coor##1,##2,##3\@nil{\polar@coor{##1}{##2}}}%
```

16 Basic graphics parameters

\psset@origin, \psk@origin

```
\def\psset@origin#1{%
  \pst@@getcoor{#1}%
  \edef\psk@origin{\pst@coor \tx@NET }}
\def\psk@origin{}
```

\psset@swapaxes, \ifpsswapaxes

```
\newif\ifswapaxes
\def\psset@swapaxes#1{%
  \@nameuse{@pst#1}%
  \if@pst
    \def\psk@swapaxes{-90 rotate -1 1 scale }%
  \else
    \def\psk@swapaxes{}%
  \fi}
\psset@swapaxes{false}
```

\psset@showpoints, \ifshowpoints

```
\newif\ifshowpoints
\def\psset@showpoints#1{\@nameuse{showpoints#1}}
\psset@showpoints{false}
```

\psset@border, \psk@border

```
\let\pst@setrepeatarrowsflag\relax
\def\psset@border#1{%
  \pst@getlength{#1}\psk@border
  \pst@setrepeatarrowsflag}
\psset@border{0pt}
```

\psset@bordercolor, \psbordercolor

```
\def\psset@bordercolor#1{\pst@getcolor{#1}\psbordercolor}
\psset@bordercolor{white}
```

\psset@doubleline, \ifpsdoubleline

```
\newif\ifpsdoubleline
\def\psset@doubleline#1{%
  \@nameuse{psdoubleline#1}%
  \pst@setrepeatarrowsflag}
\psset@doubleline{false}
```

\psset@doublesep, \psdoublesep

```
\def\psset@doublesep#1{\def\psdoublesep{#1}}
\psset@doublesep{1.25\pslinewidth}
```

\psset@doublecolor, \psdoublecolor

```
\def\psset@doublecolor#1{\pst@getcolor{#1}\psdoublecolor}
\psset@doublecolor{white}
```

\psset@shadow, \ifpsshadow

```
\newif\ifpsshadow
\def\psset@shadow#1{%
  \@nameuse{psshadow#1}%
  \pst@setrepeatarrowsflag}
\psset@shadow{false}
```

\psset@shadowsize, \psk@shadowsize

```
\def\psset@shadowsize#1{\pst@getlength{#1}\psk@shadowsize}
\psset@shadowsize{3pt}
```

\psset@shadowangle, \psk@shadowangle

```
\def\psset@shadowangle#1{\pst@getangle{#1}\psk@shadowangle}
\psset@shadowangle{-45}
```

\psset@shadowcolor, \psshadowcolor

```
\def\psset@shadowcolor#1{\pst@getcolor{#1}\psshadowcolor}
\psset@shadowcolor{darkgray}
```

\pst@setrepeatarrowsflag

```
\def\pst@repeatarrowsflag{\z@}
\def\pst@setrepeatarrowsflag{%
  \edef\pst@repeatarrowsflag{%
    \ifdim\psk@border\p@>\z@ 1\else\ifpsdoubleline 1\else
    \ifpsshadow 1\else \z@\fi\fi\fi}}
```

17 Line styles

For each `linestyle` *style*, the command `\psls@<style>` should be the PostScript code that strokes the path. The style can assume that the PostScript environment has linewidth equal to `\pslinewidth` and color equal to `\pslinecolor`.

\pst@linetype

Macros that draw lines should define `\pst@linetype` (not a count register) to be:

1,2,... A closed path whose length should be divided by `\pst@linetype` before fitting a pattern.

0 A line that has nothing at the tips.

-1 A line with an arrow or something at the end.

-2 A line with an arrow or something at the beginning.

-3 A line with an arrow or something at each end.

This information is used by the line styles that draw dashed and dotted lines in order to figure out how to adjust the patterns.

\psls@none

```
\def\psls@none{}
```

\psset@linewidth, \pslinewidth

```
\newdimen\pslinewidth
\def\psset@linewidth#1{\pssetlength\pslinewidth{#1}}
\psset@linewidth{.8pt}
```

\psset@linecolor, \pslinecolor

```
\def\psset@linecolor#1{\pst@getcolor{#1}\pslinecolor}
```

```
\psset@linecolor{black}
```

\psls@solid

```
\def\psls@solid{0 setlinecap stroke }
```

\psset@dash, \psk@dash

\psk@dash is set to the PostScript code for the dash pattern (include the trailing space).

```
\def\psset@dash#1{%
  \pst@expandafter\psset@@dash{#1} * * *\@nil
  \edef\psk@dash{\pst@number\pst@dimg \pst@number\pst@dimh}}
\def\psset@@dash#1 #2 #3\@nil{%
  \pssetlength\pst@dimg{#1}%
  \pssetlength\pst@dimh{#2}}
\psset@dash{5pt 3pt}
```

\psls@dashed

```
\def\psls@dashed{\psk@dash \pst@linetype\space \tx@DashLine}
```

\tx@DashLine

Syntax:

dim1 dim2 linetype DashLine

DashLine adjusts, and then sets, the dash pattern [<dim1 dim2>] so that it fits evenly onto a path.

```
\pst@def{DashLine}<%
  % "a" is set to the length of first and last black segment, as fraction of
  % usual black segment.
  dup 0 gt
  { /a .5 def \tx@PathLength exch div }
  { pop /a 1 def \tx@PathLength }
  ifelse
  /b ED             % Pattern should fit evenly in b
  /x ED             % Length of white segment.
  /y ED             % Length of black segment.
  /z y x add def    % Total length of dash pattern.
  % If pattern  is repeated n times, total length is (nz + 2(a-.5)y).
  % Set length to b, solve for n, round, and leave on stack:
  %   n = round((b - 2(a-.5)y)/z)
  b a .5 sub 2 mul y mul sub z \tx@Div round
  % Adjust x and y by factor k so that
  %   (n(kz) + 2(a-.5)(ky)) = b.
  % Solve for k and leave two copies on stack:
  %   k = b/(nz + 2(a-.5)y)
  z mul a .5 sub 2 mul y mul add b exch \tx@Div dup
  % Scale x and y, set dash, and stroke:
  y mul /y ED x mul /x ED
  % Make sure both x and y aren't zero:
  x 0 eq y 0 eq and { /x 1 def /y 1 def } if
  [ y x ] 1 a sub y mul setdash stroke>
```

dotsep

```
\def\psset@dotsep#1{\pst@getlength{#1}\psk@dotsep}
\psset@dotsep{3pt}
```

\psls@dotted

```
\def\psls@dotted{\psk@dotsep \pst@linetype\space \tx@DotLine}%
```

\tx@DotLine

Syntax:

dim linetype `DotLine`

`DotLine` adjusts, and then sets, the dash pattern to produce a dotted line with distance *dim* between dots so that it fits evenly onto a path. Dots are produced by setting dash pattern with length of white segment equal to distance from center of dot to center of dot, length of black segment equal to 0, and `linecap` equal to 1.

```
\pst@def{DotLine}<%
  /b \tx@PathLength def  % Path length.
  /a ED                  % \pst@linetype.
  /z ED                  % dotsep.
  /y CLW def             % linewidth (dot diameter).
  /z y z add def         % Total length of dash pattern.
  % Set b to adjusted path length that pattern should be multiple of:
  a 0 gt
    % If closed, as many dots as spaces.
    { /b b a div def }
    { a 0 eq
        % If open with no arrows, one more dot than space.
        { /b b y sub def }
        % If open one arrow, as many dots as spaces (do nothing)
        % If open two arrows, one more space than dot.
        { a -3 eq { /b b y add def } if }
      ifelse }
  ifelse
  % Let n be number of times pattern is repeated:
  %  n = round(b/z)
  % Adjust length of pattern so that it fits evenly in b:
  %  z = b/n = b/(round(b/z))
  % z is length of white segment. Length of black segment is 0.
  [ 0 b b z \tx@Div round \tx@Div dup 0 le { pop 1 } if ]
  a 0 gt                 % setting dash pattern.
    % Set offset to 0 if path is closed]
    { 0 }
    % Set offset to -(y/2) if open curve begins with arrow, (y/2) otherwise:
    { y 2 div a -2 gt { neg } if }
  ifelse
  % Setting linecap to 1 produces the dots.
  setdash 1 setlinecap stroke>
```

\psset@linestyle

```
\def\psset@linestyle#1{%
```

```
  \@ifundefined{psls@#1}%
    {\@pstrickserr{Line style `#1' not defined}\@eha}%
    {\edef\pslinestyle{#1}}}
\psset@linestyle{solid}
```

18 Fill styles

For each `fillstyle` *style*, the command `\psfs@<style>` should be the PostScript code that fills the region. The style should not assume anything about the PostScript environment's `linewidth` or color.

\psfs@none

```
\def\psfs@none{}
```

\psset@fillcolor, \psfillcolor

```
\def\psset@fillcolor#1{\pst@getcolor{#1}\psfillcolor}
\psset@fillcolor{white}
```

\psfs@solid

```
\def\psfs@solid{\pst@usecolor\psfillcolor fill }
```

hatchwidth

```
\def\psset@hatchwidth#1{\pst@getlength{#1}\psk@hatchwidth}
\psset@hatchwidth{.8pt}
```

hatchsep

```
\def\psset@hatchsep#1{\pst@getlength{#1}\psk@hatchsep}
\psset@hatchsep{4pt}
```

hatchcolor

```
\def\psset@hatchcolor#1{\pst@getcolor{#1}\pshatchcolor}
\psset@hatchcolor{black}
```

hatchangle

```
\def\psset@hatchangle#1{\pst@getangle{#1}\psk@hatchangle}
\psset@hatchangle{45}
```

\psfs@hlines

```
\def\psfs@hlines{%
  \psk@hatchangle rotate
  \psk@hatchwidth SLW
  \pst@usecolor\pshatchcolor
  \psk@hatchsep \tx@LineFill}
\@namedef{psfs@hlines*}{gsave \psfs@solid grestore \psfs@hlines}
```

\tx@LineFill

```
\pst@def{LineFill}<%
  abs CLW add /a ED              % hatchsep
  gsave
```

```
    clip
    pathbbox                    %leave llx,lly,urx,ury on stack
    a \tx@Div ceiling /y2 ED    % Number of top line to be drawn.
    /x2 ED
    a \tx@Div floor /y1 ED      % Number of bottom line to be drawn
    /x1 ED
    /n y2 y1 sub 1 add cvi def % Number of lines.
    /y1 a y1 mul def            % y-coordinate of bottom line.
    newpath 2 setlinecap
    n
    { currentstrokeadjust ==
      x1 y1 moveto
      x2 y1 L
      stroke
      /y1 y1 a add def }
    repeat
  grestore>
\pst@def{LineFill}<%
  abs CLW add /a ED              % hatchsep
  gsave
    clip
    pathbbox                    %leave llx,lly,urx,ury on stack
    a \tx@Div ceiling /y2 ED    % Number of top line to be drawn.
    /x2 ED
    a \tx@Div floor /y1 ED      % Number of bottom line to be drawn
    /x1 ED
    /n y2 y1 sub 1 add cvi def % Number of lines.
    /y1 a y1 mul def            % y-coordinate of bottom line.
    newpath 2 setlinecap
    systemdict /currentstrokeadjust known  % Level 2
    { currentstrokeadjust }
    { false }
    ifelse
    { /t { } def }
    { /t {
        transform
        0.25 sub round 0.25 add exch
        0.25 sub round 0.25 add exch
        itransform
      } bind def }
    ifelse
    n {
      x1 y1 t moveto
      x2 y1 t L
      stroke
      /y1 y1 a add def
    } repeat
  grestore>
```

\psfs@vlines

```
\def\psfs@vlines{%
  90 rotate
  \psfs@hlines}
\@namedef{psfs@vlines*}{gsave \psfs@solid grestore \psfs@vlines}
```

\psfs@crosshatch

```
\def\psfs@crosshatch{gsave \psfs@hlines grestore \psfs@vlines}
\@namedef{psfs@crosshatch*}{%
  gsave \psfs@solid grestore
  gsave \psfs@hlines grestore
  \psfs@vlines}
```

fillstyle

```
\def\psset@fillstyle#1{%
  \@ifundefined{psfs@#1}%
    {\@pstrickserr{Undefined fill style: `#1'}\@eha}%
    {\edef\psfillstyle{#1}}}
\psset@fillstyle{none}
```

19 Arrowheads and t-bars

It would be nice to use a font, with hinting.

\psset@arrows, \psk@arrowA, \psk@arrowB

`\if@pst` is used as a flag for errors.

```
\def\psset@arrows#1{%
  \begingroup
    \pst@activearrows
    \xdef\pst@tempg{#1}%
  \endgroup
  \expandafter\psset@@arrows\pst@tempg\@empty-\@empty\@nil
  \if@pst\else
    \@pstrickserr{Bad arrows specification: #1}\@ehpa
  \fi}
\def\psset@@arrows#1-#2\@empty#3\@nil{%
  \@psttrue
  \def\next##1,#1-##2,##3\@nil{\def\pst@tempg{##2}}%
  \expandafter\next\pst@arrowtable,#1-#1,\@nil
  \@ifundefined{psas@\pst@tempg}%
    {\@pstfalse\def\psk@arrowA{}}%
    {\let\psk@arrowA\pst@tempg}%
  \@ifundefined{psas@#2}%
    {\@pstfalse\def\psk@arrowB{}}%
    {\def\psk@arrowB{#2}}}
\def\psk@arrowA{}
\def\psk@arrowB{}
```

\pst@arrowtable

This is a translator for `arrowA`. Add to it with `\edef`, as in

```
\edef\pst@arrowtable{\pst@arrowtable,*o-o*}
```

```
\def\pst@arrowtable{,<->,<<->>,>-<,>>-<<,(-),[-]}
```

\pst@activearrows

This redefines certain characters in case they are active, before expanding the `arrows` argument. Add to it with `\expandafter`, as in

```
\begingroup
  \catcode`\:=13
  \expandafter\gdef\expandafter\pst@activearrows
    \expandafter{\def:{\string:}}
\endgroup
```

```
\begingroup
  \catcode`\<=13
  \catcode`\>=13
  \catcode`\|=13
  \gdef\pst@activearrows{\def<{\string<}\def>{\string>}\def|{\string|}}
\endgroup
```

BeginArrow, EndArrow

For each arrow *arrow*, `\psas@<arrow>` should be PostScript code so that

y2 x2 y1 x1 `BeginArrow \psk@arrowscale \psas@`*arrow* `EndArrow`

- Draws an arrow with the tip at *x1 y1*, and
- Leaves on the stack *y2 x2 x1' y1'*, where *x1' y1'* is the position that a connecting line should start from.

`BeginArrow` sets up an environment so that `\psas@<arrow>` only has to draw an arrow pointing down and with the tip at `0 0` and , and leave the current point where a connecting line should start from. `EndArrow` then restores the original environment and translates the current point into the original coordinate system.

A special dictionary `ADict` is used with arrows so that scratch variables will not conflict. The matrix is saved as `@mtrx` to indicate that the arrow procedures should not change this. The same is true for `@x1`, `@y1`, `@x2`, `@y2` and `@angle`, which are used by a patch of `BeginArrow` and `EndArrow` that is required for some versions of Sun's NewsPrint (see `read-me.pst`).

```
\pst@def{BeginArrow}<%
  ADict begin
  /@mtrx CM def
  gsave
    2 copy T
    2 index sub neg exch 3 index sub exch \tx@Atan
    rotate
    newpath>
\pst@def{EndArrow}<@mtrx setmatrix CP grestore end>
```

arrowscale

```
\def\psset@arrowscale#1{\pst@getscale{#1}\psk@arrowscale}
\psset@arrowscale{1}
```

\psset@arrowsize, \psk@arrowsize

```
\def\psset@arrowsize#1{%
  \pst@expandafter\pst@getdimnum{#1} {} {} {}\@nil
  \edef\psk@arrowsize{\pst@number\pst@dimg \pst@tempg}}
\psset@arrowsize{2pt 3}
```

\psset@arrowlength, \psk@arrowlength

```
\def\psset@arrowlength#1{\pst@checknum{#1}\psk@arrowlength}
\psset@arrowlength{1.4}
```

\psset@arrowinset, \psk@arrowinset

```
\def\psset@arrowinset#1{\pst@checknum{#1}\psk@arrowinset}%
\psset@arrowinset{.4}
```

\tx@Arrow

Syntax:

boolean `\psk@arrowinset \psk@arrowlength \psk@arrowsize Arrow`

boolean is `true` for reverse arrows and `false` for normal arrows.

```
\pst@def{Arrow}<%
  CLW mul add dup               % width
  2 div /w ED                   % Half width
  mul dup /h ED                 % Height
  mul /a ED                     % Inset
  { 0 h T 1 -1 scale } if   % For reverse arrows
  w neg h moveto
  0 0 L
  w h L
  w neg a neg rlineto
  gsave fill grestore>
```

\psas@>

```
\@namedef{psas@>}{%
  false \psk@arrowinset \psk@arrowlength \psk@arrowsize \tx@Arrow}
```

\psas@>>

```
\@namedef{psas@>>}{%
  false \psk@arrowinset \psk@arrowlength \psk@arrowsize \tx@Arrow
  0 h T
  gsave
    newpath
    false \psk@arrowinset \psk@arrowlength \psk@arrowsize \tx@Arrow
    CP
  grestore
  CP newpath moveto
  2 copy
  L
  stroke
  moveto}
```

\psas@<

```
\@namedef{psas@<}{%
  true \psk@arrowinset \psk@arrowlength \psk@arrowsize \tx@Arrow}
```

\psas@<<

```
\@namedef{psas@<<}{%
  true \psk@arrowinset \psk@arrowlength \psk@arrowsize \tx@Arrow
  CP newpath moveto 0 a neg L stroke 0 h neg T
  false \psk@arrowinset \psk@arrowlength \psk@arrowsize \tx@Arrow}
```

\psset@tbarsize, \psk@tbarsize

```
\def\psset@tbarsize#1{%
  \pst@expandafter\pst@getdimnum{#1} {} {} {}\@nil
  \edef\psk@tbarsize{\pst@number\pst@dimg \pst@tempg}}
\psset@tbarsize{2pt 5}
```

Tbar

Syntax

```
\psk@tbarsize Tbar
```

```
\pst@def{Tbar}<%
  CLW mul add /z ED              % width
  z -2 div CLW 2 div moveto
  z 0 rlineto
  stroke
  0 CLW moveto>
```

\psas@|

```
\@namedef{psas@|}{\psk@tbarsize \tx@Tbar}
```

\psas@|*

```
\@namedef{psas@|*}{0 CLW -2 div T \psk@tbarsize \tx@Tbar}
```

\psset@bracketlength, \psk@bracketlength

```
\def\psset@bracketlength#1{\pst@checknum{#1}\psk@bracketlength}
\psset@bracketlength{.15}
```

\tx@Bracket, \tx@@Bracket

Syntax

```
\psk@bracketlength \psk@tbarsize Bracket
```

```
\pst@def{Bracket}<%
  CLW mul add dup
  CLW sub 2 div /x ED   % adjusted half width
  mul CLW add /y ED     % y-position of height
  /z CLW 2 div def
  x neg y moveto
```

```
  x neg CLW 2 div L
  x CLW 2 div L
  x y L
  stroke
  0 CLW moveto>
```

\psas@]

```
\@namedef{psas@]}{\psk@bracketlength \psk@tbarsize \tx@Bracket}
```

\psset@rbracketlength, \psk@rbracketlength

```
\def\psset@rbracketlength#1{\pst@checknum{#1}\psk@rbracketlength}
\psset@rbracketlength{.15}
```

RoundBracket

Syntax

```
\psk@bracketlength \psk@tbarsize RoundBracket
```

```
\pst@def{RoundBracket}<%
  CLW mul add dup
  2 div /x ED           % half width
  mul /y ED             % height
  /mtrx CM def
  0 CLW 2 div T
  x y mul 0 ne { x y scale } if
  1 1 moveto
   .85 .5   .35  0   0 0 curveto
  -.35  0  -.85 .5  -1 1 curveto
  mtrx setmatrix
  stroke
  0 CLW moveto>
```

\psas@(

```
\@namedef{psas@)}{\psk@rbracketlength \psk@tbarsize \tx@RoundBracket}
```

\psas@c, \psas@cc, \psas@C

This is not going to be used frequently, and so we don't bother defining a PostScript procedure in the header.

```
\def\psas@c{1 \psas@@c}
\def\psas@cc{0 CLW 2 div T 1 \psas@@c}
\def\psas@C{2 \psas@@c}
\def\psas@@c{%
  setlinecap
  0 0 moveto
  0 CLW 2 div L
  stroke
  0 0 moveto}
```

\psas@

```
\def\psas@{}
\psset@arrows{-}
```

20 Graphics objects: processing arguments

\pst@par, \addto@par, \use@par

Graphics objects accumulate *parameter=value* pairs in the command sequence `\pst@par`. They use `\addto@par` to add to `\pst@par`, and `\use@par` to make the parameter changes effective.

```
\def\pst@par{}
\def\addto@par#1{%
  \ifx\pst@par\@empty
    \def\pst@par{#1}%
  \else
    \expandafter\def\expandafter\pst@par\expandafter{\pst@par,#1}%
  \fi}
\def\use@par{%
  \ifx\pst@par\@empty\else
    \expandafter\@psset\pst@par,\@nil
    \def\pst@par{}%
  \fi}
```

\pst@object

Any macro, such as `\psline`, that uses graphics parameters should begin as follows:

```
\def\psline{\def\pst@par{}\pst@object{psline}}
\def\psline@i{ ... }
```

`\pst@object` checkes for the optional `[<par>=<value>,...]` argument, adds key-value pairs to `\pst@par` if found, skips spaces, and then invokes `\psline@i`.

```
\def\pst@object#1{%
  \pst@ifstar{\@ifnextchar[{\pst@@object{#1}}{\@nameuse{#1@i}}}}
\def\pst@@object#1[#2]{%
  \addto@par{#2}\@ifnextchar+{\@nameuse{#1@i}}{\@nameuse{#1@i}}}
```

\newpsobject

For example,

```
\newpsobject{dottedline}{psline}{linestyle=dotted}
```

has the following effect:

```
\def\dottedline{%
  \def\pst@par{linestyle=dotted}\pst@object{psline}}
```

and thus `\dottedline` is just like `\psline`, except that the default falue of `linestyle` is changed to `dotted`.

```
\def\newpsobject#1#2#3{%
  \@ifundefined{#2@i}%
    {\@pstrickserr{Graphics object `#2' not defined}\@eha}%
    {\@namedef{#1}{\def\pst@par{#3}\pst@object{#2}}}\ignorespaces}
```

\pst@getarrows

\pst@getarrows{foo} checks for an optional argument containing arrows, and then invokes foo. The arrows argument must be followed by (.

```
\def\pst@getarrows#1{\@ifnextchar({#1}{\pst@@getarrows{#1}}}
\def\pst@@getarrows#1#2{\addto@par{arrows=#2}#1}
```

21 Graphics objects: Basics TEX macros

Each graphics object should use one of the following:

\begin@OpenObj ... \end@OpenObj	Open curves with arrows.
\begin@AltOpenObj ... \end@AltOpenObj	Open curves w/o arrows.
\begin@ClosedObj ... \end@ClosedObj	Closed curves.
\begin@SpecialObj ... \end@SpecialObj	Other.

This makes it possible for \pscustom to work by redefining these.

\begin@ClosedObj, \end@ClosedObj

```
\def\begin@ClosedObj{%
  \leavevmode
  \pst@killglue
  \begingroup
    \use@par
    \solid@star
    \ifpsdoubleline \pst@setdoublesep \fi
    \init@pscode}
\def\end@ClosedObj{%
    \ifpsshadow \pst@closedshadow \fi
    \ifdim\psk@border\p@>\z@ \pst@addborder \fi
    \pst@fill
    \pst@stroke
    \ifpsdoubleline \pst@doublestroke \fi
    \ifshowpoints
      \addto@pscode{Points aload length 2 div cvi /N ED \psdots@iii}%
    \fi
    \use@pscode
  \endgroup
  \ignorespaces}
```

\begin@OpenObj, \begin@AltOpenObj, \end@OpenObj

```
\def\begin@OpenObj{%
  \begin@ClosedObj
    \let\pst@linetype\pst@arrowtype
    \pst@addarrowdef}
```

```
\def\begin@AltOpenObj{%
  \begin@ClosedObj
    \def\pst@repeatarrowsflag{\z@}%
    \def\pst@linetype{0}}
\def\end@OpenObj{%
    \ifpsshadow \pst@openshadow \fi
    \ifdim\psk@border\p@>\z@ \pst@addborder \fi
    \pst@fill
    \pst@stroke
    \ifpsdoubleline \pst@doublestroke \fi
    \ifnum\pst@repeatarrowsflag>\z@ \pst@repeatarrows \fi
    \ifshowpoints \pst@OpenShowPoints \fi
    \use@pscode
  \endgroup
  \ignorespaces}
```

\begin@SpecialObj, \end@SpecialObj

```
\def\begin@SpecialObj{%
  \leavevmode
  \pst@killglue
  \begingroup
    \use@par
    \init@pscode}
\def\end@SpecialObj{%
    \use@pscode
  \endgroup
  \ignorespaces}
```

\init@pscode, \addto@pscode, \use@pscode

Graphics objects are built up by adding PostScript code to `\pst@code` with `\addto@pscode`. `\use@pscode` then adds leading and trailing PostScript code, and (normally) inserts it in a `\special` (it also most empty `\pst@code`). Hacks like `\psclip`, `\multips` and `\pstextpath` work by redefining `\use@pscode`. These hacks use `\use@pscode` themselves when appropriate, making limited nesting of these hacks is possible. `\PSTtoEPS` works be redefining `\addto@pscode` so that it writes to a file. All this was carefully designed so that these hacks would work. Watch out!

```
\def\pst@code{}%
\def\init@pscode{%
  \addto@pscode{%
    \pst@number\pslinewidth SLW
    \pst@usecolor\pslinecolor}}
\def\addto@pscode#1{\xdef\pst@code{\pst@code#1\space}}
\def\use@pscode{%
  \pstverb{%
    \pst@dict
      \tx@STP
      newpath
      \psk@origin
      \psk@swapaxes
      \pst@code
     end}%
   \gdef\pst@code{}}
```

\pst@killglue

```
\def\KillGlue{%
  \def\pst@killglue{\unskip\ifdim\lastskip>\z@\expandafter\pst@killglue\fi}}
\def\DontKillGlue{\let\pst@killglue\relax}
\DontKillGlue
```

\solid@star

The optional * is typically used to make a solid option. This means that `linestyle` is set to `none`, `linewidth` is set to `0`, and `fillcolor` is set to `linecolor`.

```
\def\solid@star{%
  \if@star
    \pslinewidth=\z@
    \psdoublelinefalse
    \def\pslinestyle{none}%
    \def\psfillstyle{solid}%
    \let\psfillcolor\pslinecolor
  \fi}
```

\pst@setdoublesep

```
\def\pst@setdoublesep{%
  \pst@getlength\psdoublesep\psdoublesep
  \pslinewidth=2\pslinewidth
  \advance\pslinewidth\psdoublesep\p@
  \let\pst@setdoublesep\relax}
```

\tx@Shadow

Syntax:

$x\ y$ Shadow

translates current path by $x\ y$.

```
\pst@def{Shadow}<%
  [
    { /moveto load }
    { /lineto load }
    { /curveto load }
    { /closepath load }
    pathforall
  ]
  cvx
  newpath
  3 1 roll
  T
  exec>
```

\pst@closedshadow

```
\def\pst@closedshadow{%
  \addto@pscode{%
    gsave
```

```
    \psk@shadowsize \psk@shadowangle \tx@PtoC
    \tx@Shadow
    \pst@usecolor\psshadowcolor
    gsave fill grestore
    stroke
    grestore
    gsave
    \pst@usecolor\psfillcolor
    gsave fill grestore
    stroke
    grestore}}
```

\pst@openshadow

```
\def\pst@openshadow{%
  \addto@pscode{%
    gsave
      \psk@shadowsize \psk@shadowangle \tx@PtoC
      \tx@Shadow
      \pst@usecolor\psshadowcolor
      \ifx\psfillstyle\@none\else
        gsave fill grestore
      \fi
      stroke}%
  \pst@repeatarrows
  \addto@pscode{grestore}
  \ifx\psfillstyle\@none\else
    \addto@pscode{%
      gsave
      \pst@usecolor\psfillcolor
      gsave fill grestore
      stroke
      grestore}
   \fi}
```

\pst@addborder

```
\def\pst@addborder{%
  \addto@pscode{%
    gsave
      \psk@border 2 mul
      CLW add SLW
      \pst@usecolor\psbordercolor
      stroke
    grestore}}
```

\pst@stroke

```
\def\pst@stroke{%
  \ifx\pslinestyle\@none\else
    \addto@pscode{%
      gsave
        \pst@number\pslinewidth SLW
        \pst@usecolor\pslinecolor
        \@nameuse{psls@\pslinestyle}
      grestore}%
```

```
  \fi}
```

\pst@fill

```
\def\pst@fill{%
  \ifx\psfillstyle\@none\else
    \addto@pscode{gsave \@nameuse{psfs@\psfillstyle} grestore}%
  \fi}
```

\pst@doublestroke

```
\def\pst@doublestroke{%
  \addto@pscode{%
    gsave
      \psdoublesep SLW
      \pst@usecolor\psdoublecolor
      stroke
    grestore}}
```

\pst@arrowtype

```
\def\pst@arrowtype{%
  \ifx\psk@arrowB\@empty 0 \else -2 \fi
  \ifx\psk@arrowA\@empty 0 \else -1 \fi
  add}
```

\pst@addarrowdef, \pst@arrowdef, \pst@arrowtype

`ArrowA` takes two coordinates from the stack, draws the arrow with the tip at the top coordinate, leaves the second coordinate on the stack and leaves the current point where a line should join.

`ArrowB` takes two coordinates from the stack, draws the arrow with the tip at the top coordinate, and leaves both coordinates on the stack, without changing the graphics state.

This particular definition of `ArrowA` is important for `\pscustom`.

`ArrowA` and `ArrowB` might also save the arrow coordinates, because sometimes it is necessary to redraw the arrows (see `\pst@setrepeatarrowsflag`).

```
\def\pst@addarrowdef{%
  \addto@pscode{%
    /ArrowA {
      \ifx\psk@arrowA\@empty
        \pst@oplineto
      \else
        \pst@arrowdef{A}
        moveto
      \fi
    } def
    /ArrowB {
      \ifx\psk@arrowB\@empty \else \pst@arrowdef{B} \fi
    } def}}
\def\pst@arrowdef#1{%
  \ifnum\pst@repeatarrowsflag>\z@
    /Arrow#1c [ 6 2 roll ] cvx def Arrow#1c
  \fi
  \tx@BeginArrow
```

```
  \psk@arrowscale
  \@nameuse{psas@\@nameuse{psk@arrow#1}}
  \tx@EndArrow}
```

\pst@repeatarrows

```
\def\pst@repeatarrows{%
  \addto@pscode{%
    gsave
    \ifx\psk@arrowA\@empty\else
      ArrowAc ArrowA pop pop
    \fi
    \ifx\psk@arrowB\@empty\else
      ArrowBc ArrowB pop pop pop pop
    \fi
    grestore}}
```

\pst@OpenShowPoints

```
\def\pst@OpenShowPoints{%
  \addto@pscode{%
    gsave
      \psk@dotsize
      \@nameuse{psds@\psk@dotstyle}
      /TheDot {
        gsave T \psk@dotangle \psk@dotscale Dot grestore
      } def
      newpath
      Points aload length 2 div 2 sub cvi /N ED
      N 0 ge
      { \ifx\psk@arrowA\@empty
          TheDot
        \else
          pop pop
        \fi
        N { TheDot } repeat
        \ifx\psk@arrowB\@empty
          TheDot
        \else
          pop pop
        \fi }
      { N 2 mul { pop } repeat }
      ifelse
    grestore}}
```

22 Custom graphics

Graphics objects using `\begin@SpecialObj` cannot be used with `\pscustom`. It is up to the other graphics objects to be compatible with `\pscustom`. This means:

- To use the current point as an additional coordinate, when it exists, the graphics object should insert `\pst@cp`.
- For graphics objects that use `\begin@OpenObj`, the `ArrowA` is defined by `\pscustom` to connect the top coordinate with the current point by a line, if

there is a current point. Other graphics objects should use `\pst@oplineto` as a substitute for `moveto` if they wish to connect a coordinate to the current point if it exists.

Closed graphics objects are not under an obligation to anything particularly sensible the current point exists.

\pscustom

The main graphics object modifies `\begin@OpenObj` and `\end@OpenObj` so that the open curves extend the current path.

```
\def\pscustom{\def\pst@par{}\pst@object{pscustom}}
\long\def\pscustom@i#1{%
  \begin@SpecialObj
    \solid@star
    \let\pst@ifcustom\iftrue
    \let\begin@ClosedObj\begin@CustomObj
    \let\end@ClosedObj\endgroup
    \def\begin@OpenObj{\begin@CustomObj\pst@addarrowdef}%
    \let\end@OpenObj\endgroup
    \let\begin@AltOpenObj\begin@CustomObj
    \def\begin@SpecialObj{%
      \begingroup
      \pst@misplaced{special graphics object}%
      \def\addto@pscode####1{}
      \let\end@SpecialObj\endgroup}%
    \def\@multips(##1)(##2)##3##4{\pst@misplaced\multips}%
    \def\psclip##1{\pst@misplaced\psclip}%
    \def\pst@repeatarrowsflag{\z@}%
    \let\pst@setrepeatarrowsflag\relax
    \showpointsfalse
    \let\showpointstrue\relax
    \def\pst@linetype{\pslinetype}%
    \let\psset@liftpen\psset@@liftpen
    \psset@liftpen{\z@}%
    \def\pst@cp{/currentpoint load stopped pop }%
    \def\pst@oplineto{/lineto load stopped { moveto } if }%
    \def\pst@optcp##1##2{%
      \ifnum##1=\z@\def##2{/currentpoint load stopped { 0 0 } if }\fi}%
    \let\caddto@pscode\addto@pscode
    \def\cuse@par##1{{\use@par##1}}%
    \the\pst@customdefs
    \setbox\pst@hbox=\hbox{#1}%
    \pst@fill
    \pst@stroke
  \end@SpecialObj}
```

\begin@CustomObj, \end@CustomObj

```
\def\begin@CustomObj{%
  \begingroup
    \use@par
    \addto@pscode{%
```

```
      \pst@number\pslinewidth SLW
      \pst@usecolor\pslinecolor}}
```

\psset@liftpen, \pst@cp, \pst@oplineto, \pst@optcp

```
\def\pst@oplineto{moveto }
\def\pst@cp{}
\def\pst@optcp#1#2{}
\def\psset@liftpen#1{}
\def\psset@@liftpen#1{%
  \ifcase#1\relax
    \def\psk@liftpen{\z@}%
    \def\pst@cp{/currentpoint load stopped pop }%
    \def\pst@oplineto{/lineto load stopped { moveto } if }%
  \or
    \def\psk@liftpen{1}%
    \def\pst@cp{}%
    \def\pst@oplineto{/lineto load stopped { moveto } if }%
  \or
    \def\psk@liftpen{2}%
    \def\pst@cp{}%
    \def\pst@oplineto{moveto }%
  \fi}
\psset@liftpen{0}
\def\psk@liftpen{-1}
```

\psset@linetype, \pslinetype

```
\def\psset@linetype#1{%
  \pst@getint{#1}\pslinetype
  \ifnum\pst@dimg<-3
    \@pstrickserr{linetype must be greater than -3}\@ehpa
    \def\pslinetype{0}%
  \fi}
\psset@linetype{0}
```

\caddto@pscode

Commands that should only occur in \pscustom should use this. Obsolete?

```
\def\caddto@pscode#1{%
  \@pstrickserr{Command can only be used in \string\pscustom}\@ehpa}
\let\cuse@par\caddto@pscode
```

\tx@MSave, \tx@MRestore

It doesn't seem worth adding these to the header file.

```
\def\tx@MSave{%
  /msavemtrx
  [ tx@Dict /msavemtrx known { msavemtrx aload pop } if CM ]
  def }
\def\tx@MRestore{%
  tx@Dict /msavemtrx known { length 0 gt } { false } ifelse
  { /msavematrx [ msavematrx aload pop setmatrix ] def }
  if }
```

\psmove, \psclosepath, \psgroup

```
\newtoks\pst@customdefs
\pst@customdefs{%
  \def\newpath{\addto@pscode{newpath}}%
  \def\moveto(#1){\pst@@getcoor{#1}\addto@pscode{\pst@coor moveto}}%
  \def\closepath{\addto@pscode{closepath}}%
  \def\gsave{\begingroup\addto@pscode{gsave}}%
  \def\grestore{\endgroup\addto@pscode{grestore}}%
  \def\translate(#1){\pst@@getcoor{#1}\addto@pscode{\pst@coor moveto}}%
  \def\rotate#1{\pst@@getangle{#1}\addto@pscode{\pst@angle rotate}}%
  \def\scale#1{\pst@getscale{#1}\pst@tempg\addto@pscode{\pst@tempg}}%
  \def\msave{\addto@pscode{\tx@MSave}}%
  \def\mrestore{\addto@pscode{\tx@MRestore}}%
  \def\swapaxes{\addto@pscode{-90 rotate -1 1 scale}}%
  \def\stroke{\def\pst@par{}\pst@object{stroke}}%
  \def\fill{\def\pst@par{}\pst@object{fill}}%
  \def\openshadow{\def\pst@par{}\pst@object{openshadow}}%
  \def\closedshadow{\def\pst@par{}\pst@object{closedshadow}}%
  \def\movepath(#1){\pst@@getcoor{#1}\addto@pscode{\pst@coor tx@Shadow}}%
  \def\lineto{\pst@onecoor{lineto}}%
  \def\rlineto{\pst@onecoor{rlineto}}%
  \def\curveto{\pst@threecoor{curveto}}%
  \def\rcurveto{\pst@threecoor{rcurveto}}%
  \def\code#1{\addto@pscode{#1}}%
  \def\coor(#1){\pst@@getcoor{#1}\addto@pscode\pst@coor\@ifnextchar({\coor}{}}%
  \def\rcoor{\pst@getcoors{}{}}%
  \def\dim#1{\pssetlength\pst@dimg{#1}\addto@pscode{\pst@number\pst@dimg}}%
  \def\setcolor#1{%
    \@ifundefined{color@#1}{}{\addto@pscode{\use@color{#1}}}}%
  \def\arrows#1{{\psset@arrows{#1}\pst@addarrowdef}}%
  \let\file\pst@rawfile
} % END \pst@customdefs
\def\closedshadow@i{\cuse@par\pst@closedshadow}
\def\openshadow@i{\cuse@par\pst@openshadow}
\def\stroke@i{\cuse@par\pst@stroke}%
\def\fill@i{\cuse@par\pst@fill}%
\def\pst@onecoor#1(#2){%
  \pst@@getcoor{#2}%
  \addto@pscode{\pst@coor #1}}
\def\pst@threecoor#1(#2)#3(#4)#5(#6){%
  \begingroup
    \pst@getcoor{#2}\pst@tempa
    \pst@getcoor{#4}\pst@tempb
    \pst@getcoor{#6}\pst@tembc
    \addto@pscode{\pst@tempa \pst@tempb \pst@tempc #1}%
  \endgroup}
```

\psrawfile, \pst@rawfile

```
\def\pst@rawfile#1{%
  \begingroup
    \def\do##1{\catcode`##1=12\relax}"
    \dospecials
    \catcode`\%=14
```

```
    \pst@@rawfile{#1}%
  \endgroup}
\def\pst@@rawfile#1{%
  \immediate\openin1 #1
  \ifeof1
    \@pstrickserr{File `#1' not found}\@ehpa
  \else
    \immediate\read1 to \pst@tempg
    \loop
      \ifeof1 \@pstfalse\else\@psttrue\fi
    \if@pst
      \addto@pscode\pst@tempg
      \immediate\read1 to \pst@tempg
    \repeat
  \fi
  \immediate\closein1\relax}
```

23 Graphics objects: Basic PostScript macros

SD

```
\pst@def{SD}<%
  0 360 arc fill>
\pst@def{SQ}<%
  /r ED
  r r moveto
  r r neg L
  r neg r neg L
  r neg r L
  fill>
\pst@def{ST}<%
  /y ED /x ED
  x y moveto
  x neg y L
  0 x L
  fill>
\pst@def{SP}<%
  /r ED
  gsave
    0 r moveto
    4 { 72 rotate 0 r L } repeat
    fill
  grestore>
\@namedef{psds@*}{/Dot { 0 0 DS \tx@SD } def}
\@namedef{psds@o}{%
  /r2 DS CLW sub def
  /Dot { 0 0 DS \tx@SD \pst@usecolor\psfillcolor 0 0 r2 \tx@SD } def}
\@namedef{psds@square*}{%
  /r1 DS .886 mul def
  /Dot { r1 \tx@SQ } def}
\@namedef{psds@square}{%
  /r1 DS .886 mul def /r2 r1 CLW sub def
  /Dot { r1 \tx@SQ \pst@usecolor\psfillcolor r2 \tx@SQ } def}
\@namedef{psds@triangle*}{%
```

```
  /y1 DS .778 mul neg def /x1 y1 1.732 mul neg def
  /Dot { x1 y1 \tx@ST } def}
\@namedef{psds@triangle}{%
  /y1 DS .778 mul neg def /x1 y1 1.732 mul neg def
  /y2 y1 CLW add def /x2 y2 1.732 mul neg def
  /Dot { x1 y1 \tx@ST \pst@usecolor\psfillcolor x2 y2 \tx@ST } def}
\@namedef{psds@pentagon*}{%
  /r1 DS 1.149 mul def
  /Dot { r1 \tx@SP } def}
\@namedef{psds@pentagon}{%
  DS .93 mul dup 1.236 mul /r1 ED CLW sub 1.236 mul /r2 ED
  /Dot { r1 \tx@SP \pst@usecolor\psfillcolor
    r2 \tx@SP } def}
\@namedef{psds@+}{%
  /DS DS 1.253 mul def
  /Dot { DS 0 moveto DS neg 0 L stroke
    0 DS moveto 0 DS neg L stroke } def}
\@namedef{psds@|}{%
  \psk@tbarsize CLW mul add 2 div /DS ED
  /Dot { 0 DS moveto 0 DS neg L stroke } def}
```

dotstyle

```
\def\psset@dotstyle#1{%
  \@ifundefined{psds@#1}%
    {\@pstrickserr{Dot style `#1' not defined}\@eha}%
    {\edef\psk@dotstyle{#1}}}
\psset@dotstyle{*}
```

NArray

Syntax:

array of points NArray *points*

Sets `n` to the number of pairs in the array, and makes sure there is an even number of elements.

```
\pst@def{NArray}<%
  aload length 2 div dup
  dup cvi eq not { exch pop } if
  /n exch cvi def>
\pst@def{NArray}<%
  /f ED
  counttomark 2 div
  dup cvi /n ED
  n eq not { exch pop } if
  f
    { ] aload /Points ED }
    { n 2 mul 1 add -1 roll pop }
  ifelse>
```

Line

Syntax:

array of points Line -

`ArrowA` and `ArrowB` should be defined to draw arrows, and `Lineto` should be the procedure used to draw the path; either `lineto` or `Arcto`.

```
\pst@def{Line}<%
  \tx@NArray
  n 0 eq not
  { n 1 eq { 0 0 /n 2 def } if
    ArrowA
    /n n 2 sub def
    n { Lineto } repeat
    CP 4 2 roll ArrowB L
    pop pop }
  if>
```

Arcto

Syntax:

x2 y2 x1 y1 Arcto *x2 y2*

`r` should be set to the arc radius. Adds to the path with `arcto`, with the corner at *x1 y1* and going towards *x2 y2*. Works even when the points are equal. For use with `Line` and `Polygon`.

```
\pst@def{Arcto}<%
  /a [ 6 -2 roll ] cvx def
  a r /arcto load stopped { 5 } { 4 } ifelse { pop } repeat a>
```

Polygon

Syntax:

array of points Line -

`Lineto` should be the procedure used to draw the path; either `lineto` or `Arcto`.

```
\pst@def{CheckClosed}<%
  dup n 2 mul 1 sub index eq 2 index n 2 mul 1 add index eq and
  { pop pop /n n 1 sub def }
  if>
\pst@def{Polygon}<%
  \tx@NArray
  n 2 eq { 0 0 /n 3 def } if
  n 3 lt
  { n { pop pop } repeat }
  { n 3 gt { \tx@CheckClosed } if
    n 2 mul -2 roll /y0 ED /x0 ED
    /y1 ED /x1 ED x1 y1
    /x1 x0 x1 add 2 div def
    /y1 y0 y1 add 2 div def
    x1 y1 moveto
    /n n 2 sub def
    n { Lineto } repeat
    x1 y1 x0 y0 6 4 roll
    Lineto Lineto pop pop closepath }
  ifelse>
```

24 Interpolated curves

This documentation is largely junk.

There one was an alternate algorithm that had the nice property that when the coordinates were scaled, the interpolated curve would scale in the same way. It was also simpler. However, this one gives nicer looking results in most cases.

Two parameters should be defined:

a Lower values make the curve tighter. (Default: 1)

b Higher values make the curve tighter where the angle ABC is less than 45 degrees, and loosen the curve elsewhere. (Default: .1)

`ArrowA` and `ArrowB` should be defined as well.

Each two points are connected by a single Bezier curve, using curveto. For each point P, let P- and P+ be the control points before and after the point. I.e., If A, B and C are consecutive points, then A and B are connected by the Bezier curve with control points A, A+, B- and B, and B and C are connected with control points B, B+, C- and C.

The interpolation is local, meaning that control points B- and B+ depend only on points A, B and C.

`\tx@CCA`, `\tx@CC`

The first three lines before `CCA` set `x1`= $Ax+$, `y1`= $Ay+$, `l0`= $d(A,B)$, `dx0`= $Bx - Ax$, and `dy0`= $By - Ay$. After `CCA`, `x`= Bx, `y`= By, `dx1`= $Cx - Bx$, `dy2`= $Cy - By$, and `l1`= $d(B,C)$.

```
\pst@def{CCA}<%
  /y ED /x ED 2 copy
  y sub /dy1 ED x sub /dx1 ED
  /l1 dx1 dy1 \tx@Pyth def>
\pst@def{CCA}<%
  /y ED /x ED 2 copy
  y sub /dy1 ED x sub /dx1 ED
  /l1 dx1 dy1 \tx@Pyth def>
\pst@def{CC}<%
  /l0 l1 def
  /x1 x dx sub def /y1 y dy sub def
  /dx0 dx1 def /dy0 dy1 def
  \tx@CCA
```

The task is now to calculate $B-$ and $B+$. We first calculate the slope `dx` and `dy` at B. This tangent at B should be perpendicular to the bisection of the angle ABC. Recalling that `dx0` and `dy0` "point" from A to B, this tangency thus passes through $B + (dx, dy)$, where (dx, dy) is the average of $dx0, dy0$ and $dx1, dy1$, once these have been normalized to have the same length. If we normalize by dividing each by their length, and then multiplying both by both lengths, we get

$$\begin{aligned} \texttt{dx} &= \texttt{l1} \times \texttt{dx0} + \texttt{l0} \times \texttt{dx1} \\ \texttt{dy} &= \texttt{l1} \times \texttt{dy0} + \texttt{l0} \times \texttt{dy1} \end{aligned}$$

```
  /dx dx0 l1 c exp mul dx1 l0 c exp mul add def
  /dy dy0 l1 c exp mul dy1 l0 c exp mul add def
```

dx and dy give us the direction of the control points $B-$ and $B+$ from B. Now we adjust the distance of these control points. The first component is sine of the angle ABC, so that smaller angles give closer control points. This is raised to the b, so that b controls the extent of this dependency (and can even reverse the relation). Then this amount is multiplied times a, which those adjusts the overall tightness, independently of the angle. Let's call this amount M. This amount is then divided by the length of the vector (dx, dy), thereby normalizing this vector to unit length, and then, multiplied times the distance between A and B (for calculating $B-$). Thus, $B-$ is distance $Md(A, B)$ from B. $(x2, y2)$ are set to $B-$, thus calculated, and $B+$ is temporily stored in (dx, dy).

```
  /m dx0 dy0 \tx@Atan dx1 dy1 \tx@Atan sub
     2 div cos abs b exp a mul
     dx dy \tx@Pyth \tx@Div 2 div def
  /x2 x l0 dx mul m mul sub def
  /y2 y l0 dy mul m mul sub def
  /dx l1 dx mul m mul neg def
  /dy l1 dy mul m mul neg def>
```

\tx@IC, \tx@BOC, \tx@NC, \tx@EOC, \tx@BAC, \tx@NAC, \tx@EAC

These are the compenents of the loops that go through the lists of points that are to be interpolated. These are abbreviations, as follows:

IC	Initialize Curve
BOC	Begin Open Curve
NC	Next Curve
EOC	End Open Curve
BAC	Begin Alternative Curve
NAC	Next Alternative Curve
EAC	End Alternative Curve

```
\pst@def{IC}<%
  /c c 1 add def
  c 0 lt { /c 0 def } { c 3 gt { /c 3 def } if } ifelse
  /a a 2 mul 3 div 45 cos b exp div def
  \tx@CCA /dx 0 def /dy 0 def>
\pst@def{BOC}<%
  \tx@IC \tx@CC x2 y2 x1 y1 ArrowA
  CP 4 2 roll x y curveto>
\pst@def{NC}<\tx@CC x1 y1 x2 y2 x y curveto>
\pst@def{EOC}<%
  x dx sub y dy sub 4 2 roll ArrowB 2 copy curveto>
\pst@def{BAC}<%
  \tx@IC \tx@CC x y moveto \tx@CC
  x1 y1 CP ArrowA>
\pst@def{NAC}<x2 y2 x y curveto \tx@CC x1 y1>
\pst@def{EAC}<x2 y2 x y ArrowB curveto pop pop>
```

OpenCurve

Syntax:

array of points OpenCurve

```
\pst@def{OpenCurve}<%
  \tx@NArray
  n 3 lt
  { n { pop pop } repeat }
  { \tx@BOC
    /n n 3 sub def
    n { \tx@NC } repeat
    \tx@EOC }
  ifelse>
```

AltCurve

Syntax:

array of points AltCurve

```
\pst@def{AltCurve}<%
  { false \tx@NArray
    n 2 mul 2 roll
    [ n 2 mul 3 sub 1 roll ]
    aload /Points ED
    n 2 mul -2 roll }
  { false \tx@NArray }
  ifelse
  n 4 lt
  { n { pop pop } repeat }
  { \tx@BAC
    /n n 4 sub def
    n { \tx@NAC } repeat
    \tx@EAC }
  ifelse>
```

ClosedCurve

Syntax:

array of points ClosedCurve

```
\pst@def{ClosedCurve}<%
  \tx@NArray
  n 3 lt
  { n { pop pop } repeat }
  { n 3 gt { \tx@CheckClosed } if
    6 copy n 2 mul 6 add 6 roll
    \tx@IC \tx@CC x y moveto
    n { \tx@NC } repeat
   closepath pop pop }
  ifelse>
```

curvature

```
\def\psset@curvature#1{%
  \edef\pst@tempg{#1 }%
```

```
  \expandafter\psset@@curvature\pst@tempg * * * \@nil}
\def\psset@@curvature#1 #2 #3 #4\@nil{%
  \pst@checknum{#1}\pst@tempg
  \pst@checknum{#2}\pst@temph
  \pst@checknum{#3}\pst@tempi
  \edef\psk@curvature{\pst@tempg \pst@temph \pst@tempi}}
\psset@curvature{1 .1 0}
```

\pscurve

```
\def\pscurve{\def\pst@par{}\pst@object{pscurve}}
\def\pscurve@i{%
  \pst@getarrows{%
    \begin@OpenObj
      \pst@getcoors[\pscurve@ii}}
\def\pscurve@ii{%
  \addto@pscode{%
    \pst@cp
    \psk@curvature\space /c ED /b ED /a ED
    \ifshowpoints true \else false \fi
    \tx@OpenCurve}%
  \end@OpenObj}
```

\psecurve

```
\def\psecurve{\def\pst@par{}\pst@object{psecurve}}
\def\psecurve@i{%
  \pst@getarrows{%
    \begin@OpenObj
    \pst@getcoors[\psecurve@ii}}
\def\psecurve@ii{%
  \addto@pscode{%
    \psk@curvature\space /c ED /b ED /a ED
    \ifshowpoints true \else false \fi
    \tx@AltCurve}%
  \end@OpenObj}
```

\psccurve

```
\def\psccurve{\def\pst@par{}\pst@object{psccurve}}
\def\psccurve@i{%
  \begin@ClosedObj
    \pst@getcoors[\psccurve@ii}
\def\psccurve@ii{%
  \addto@pscode{%
    \psk@curvature\space /c ED /b ED /a ED
    \ifshowpoints true \else false \fi
    \tx@ClosedCurve}%
  \def\pst@linetype{1}%
  \end@ClosedObj}
```

25 Dots

It would be nice to use a font, with hinting.

dotsize

```
\def\psset@dotsize#1{%
  \edef\pst@tempg{#1 }%
  \expandafter\psset@@dotsize\pst@tempg -1 -1 -1\@nil}
\def\psset@@dotsize#1 #2 #3\@nil{%
  \pst@checknum{#2}\pst@tempg
  \pssetlength\pst@dimg{#1}%
  \edef\psk@dotsize{%
    /DS \pst@number\pst@dimg \pst@tempg CLW mul add 2 div def }}
\psset@dotsize{.5pt 2.5}
```

\psset@dotscale

```
\def\psset@dotscale#1{\pst@getscale{#1}\psk@dotscale}
\psset@dotscale{1}
```

\pst@Getangle

```
\def\pst@Getangle#1#2{%
  \pst@getangle{#1}\pst@tempg
  \def\pst@temph{0. }%
  \ifx\pst@tempg\pst@temph
    \def#2{}%
  \else
    \edef#2{\pst@tempg\space rotate }%
  \fi}
```

dotangle

```
\def\psset@dotangle#1{\pst@Getangle{#1}\psk@dotangle}
\psset@dotangle{0}
```

\psdots

```
\def\psdots{\def\pst@par{}\pst@object{psdots}}
\def\psdots@i{%
  \begin@SpecialObj
    \pst@getcoors[\psdots@ii}
\def\psdots@ii{%
   \addto@pscode{false \tx@NArray \psdots@iii}%
 \end@SpecialObj}
\def\psdots@iii{%
  \psk@dotsize
  \@nameuse{psds@\psk@dotstyle}
  newpath
  n { gsave T \psk@dotangle \psk@dotscale Dot grestore } repeat}
```

EndDot

Syntax

{*fill*} {*displace*} EndDot

DS should be defined to be the dot size.

```
\pst@def{EndDot}<%
```

```
  { /z DS def } { /z 0 def } ifelse
  /b ED
  0 z DS \tx@SD
  b { 0 z DS CLW sub \tx@SD } if
  0 DS z add CLW 4 div sub moveto>
```

\psas@oo

```
\def\psas@oo{{\pst@usecolor\psfillcolor true} true \psk@dotsize \tx@EndDot}
```

\psas@o

```
\def\psas@o{{\pst@usecolor\psfillcolor true} false \psk@dotsize \tx@EndDot}
```

\psas@**

```
\@namedef{psas@**}{{false} true \psk@dotsize \tx@EndDot}
```

\psas@*

```
\@namedef{psas@*}{{false} false \psk@dotsize \tx@EndDot}
```

26 Lines and polygons

linearc

```
\newdimen\pslinearc
\def\psset@linearc#1{\pssetlength\pslinearc{#1}}
\psset@linearc{0pt}
```

\psline

```
\def\psline{\def\pst@par{}\pst@object{psline}}
\def\psline@i{%
  \pst@getarrows{%
    \begin@OpenObj
      \pst@getcoors[\psline@ii}}
\def\psline@ii{%
    \addto@pscode{\pst@cp \psline@iii \tx@Line}%
  \end@OpenObj}
\def\psline@iii{%
  \ifdim\pslinearc>\z@
    /r \pst@number\pslinearc def
    /Lineto { \tx@Arcto } def
  \else
    /Lineto /lineto load def
  \fi
  \ifshowpoints true \else false \fi}
```

\qline

```
\def\qline(#1)(#2){%
  \def\pst@par{}%
  \begin@SpecialObj
    \def\pst@linetype{0}%
    \pst@getcoor{#1}\pst@tempa
    \pst@@getcoor{#2}%
```

```
    \addto@pscode{%
      \pst@tempa moveto \pst@coor L
      \@nameuse{psls@\pslinestyle}}%
  \end@SpecialObj}
```

\pspolygon

```
\def\pspolygon{\def\pst@par{}\pst@object{pspolygon}}
\def\pspolygon@i{%
  \begin@ClosedObj
    \def\pst@cp{}%
    \pst@getcoors[\pspolygon@ii}
\def\pspolygon@ii{%
    \addto@pscode{\psline@iii \tx@Polygon}%
    \def\pst@linetype{1}%
  \end@ClosedObj}
```

framearc

```
\def\psset@framearc#1{\pst@checknum{#1}\psk@framearc}
\psset@framearc{0}
```

cornersize

```
\def\psset@cornersize#1{%
  \pst@expandafter\psset@@cornersize{#1}\@nil}
\def\psset@@cornersize#1#2\@nil{%
  \if #1a\relax
    \def\psk@cornersize{\pst@number\pslinearc false }%
  \else
    \def\psk@cornersize{\psk@framearc true }%
  \fi}
\psset@cornersize{relative}
```

Frame

Syntax

framearc/linearc bool x1 y1 x2 y2 dimen `Frame`

```
\pst@def{Rect}<%
  x1 y1 y2 add 2 div moveto
  x1 y2 lineto
  x2 y2 lineto
  x2 y1 lineto
  x1 y1 lineto
  closepath>
\pst@def{OvalFrame}<%
  x1 x2 eq y1 y2 eq or
  { pop pop x1 y1 moveto x2 y2 L }
  { y1 y2 sub abs x1 x2 sub abs
    2 copy gt { exch pop } { pop } ifelse
    2 div
    exch   % STACK: cornersize halfwidth boolean
    { dup 3 1 roll mul exch }
    if
```

```
    2 copy lt { pop } { exch pop } ifelse
    /b ED
    x1 y1 y2 add 2 div moveto
    x1 y2 x2 y2 b arcto
    x2 y2 x2 y1 b arcto
    x2 y1 x1 y1 b arcto
    x1 y1 x1 y2 b arcto
    16 { pop } repeat
    closepath }
  ifelse>
\pst@def{Frame}<%
  CLW mul /a ED
  3 -1 roll 2 copy gt { exch } if
  a sub /y2 ED a add /y1 ED
  2 copy gt { exch } if
  a sub /x2 ED a add /x1 ED
  1 index 0 eq { pop pop \tx@Rect } { \tx@OvalFrame } ifelse>
```

dimen

```
\def\psset@dimen#1{%
  \pst@expandafter\psset@@dimen{#1}\@nil}
\def\psset@@dimen#1#2\@nil{%
  \if #1o\relax
    \def\psk@dimen{.5 }%
  \else
    \if #1m\relax
      \def\psk@dimen{0 }%
    \else
      \if #1i\relax
        \def\psk@dimen{-.5 }%
      \fi
    \fi
  \fi}
\psset@dimen{outer}
```

\psframe

```
\def\psframe{\def\pst@par{}\pst@object{psframe}}
\def\psframe@i(#1){%
  \@ifnextchar({\psframe@ii(#1)}{\psframe@ii(0,0)(#1)}}
\def\psframe@ii(#1)(#2){%
  \begin@ClosedObj
    \pst@getcoor{#1}\pst@tempa
    \pst@@getcoor{#2}%
    \addto@pscode{\psk@cornersize \pst@tempa \pst@coor \psk@dimen \tx@Frame}%
    \def\pst@linetype{2}%
    \showpointsfalse
  \end@ClosedObj}
```

27 Curves

\psbezier

```
\def\psbezier{\def\pst@par{}\pst@object{psbezier}}
\def\psbezier@i{\pst@getarrows\psbezier@ii}
\def\psbezier@ii#1(#2)#3(#4)#5(#6){%
  \@ifnextchar({\psbezier@iii{1}(#2)(#4)(#6)}%
    {\psbezier@iii{\z@}(0,0)(#2)(#4)(#6)}}
\def\psbezier@iii#1(#2)(#3)(#4)(#5){%
  \begin@OpenObj
  \pst@getcoor{#2}\pst@tempa
  \pst@getcoor{#3}\pst@tempb
  \pst@getcoor{#4}\pst@tempc
  \pst@getcoor{#5}\pst@tempd
  \pst@optcp{#1}\pst@tempa
  \ifshowpoints\psbezier@iv\fi
  \addto@pscode{
    \pst@tempb \pst@tempa ArrowA
    \pst@tempc \pst@tempd ArrowB
    curveto}%
  \end@OpenObj}
\def\psbezier@iv{%
  \addto@pscode{%
    gsave
      \pst@tempa \pst@tempb \pst@tempc \pst@tempd
       newpath moveto L L L
       CLW 2 div SLW
       [ \psk@dash\space ] 0 setdash stroke
    grestore
    /Points [\pst@tempa\pst@tempb\pst@tempc\pst@tempd] def}}
```

\parabola

```
\pst@def{Parab}<%
  /y0 exch def
  /x0 exch def
  /y1 exch def
  /x1 exch def
  /dx x0 x1 sub 3 div def
  /dy y0 y1 sub 3 div def
  x0 dx sub y0 dy add x1 y1 ArrowA
  x0 dx add y0 dy add x0 2 mul x1 sub y1 ArrowB
  curveto
  /Points [ x1 y1 x0 y0 x0 2 mul x1 sub y1 ] def>
\def\parabola{\def\pst@par{}\pst@object{parabola}}
\def\parabola@i{\pst@getarrows\parabola@ii}
\def\parabola@ii#1(#2)#3(#4){%
  \begin@OpenObj
    \pst@getcoor{#2}\pst@tempa
    \pst@@getcoor{#4}%
    \addto@pscode{\pst@tempa \pst@coor \tx@Parab}%
  \end@OpenObj}
```

28 Grids

gridwidth

```
\def\psset@gridwidth#1{\pst@getlength{#1}\psk@gridwidth}
```

```
\psset@gridwidth{.8pt}
```

griddots

```
\def\psset@griddots#1{%
  \pst@cntg=#1\relax
  \edef\psk@griddots{\the\pst@cntg}}
\psset@griddots{0}
```

gridcolor

```
\def\psset@gridcolor#1{\pst@getcolor{#1}\psgridcolor}
\psset@gridcolor{black}
```

subgridwidth

```
\def\psset@subgridwidth#1{\pst@getlength{#1}\psk@subgridwidth}
\psset@subgridwidth{.4pt}
```

subgridcolor

```
\def\psset@subgridcolor#1{\pst@getcolor{#1}\pssubgridcolor}
\psset@subgridcolor{gray}
```

subgriddots

```
\def\psset@subgriddots#1{%
  \pst@cntg=#1\relax\edef\psk@subgriddots{\the\pst@cntg}}
\psset@subgriddots{0}
```

subgriddiv

```
\def\psset@subgriddiv#1{%
  \pst@cntg=#1\relax\edef\psk@subgriddiv{\the\pst@cntg}}
\psset@subgriddiv{5}
```

gridlabels

```
\def\psset@gridlabels#1{\pst@getlength{#1}\psk@gridlabels}
\psset@gridlabels{10pt}
```

gridlabelcolor

```
\def\psset@gridlabelcolor#1{\pst@getcolor{#1}\psgridlabelcolor}
\psset@gridlabelcolor{black}
```

Grid

Syntax:

> *x1 y1 x2 y2 x-origin y-origin x-divsize y-divsize*
> *numsubdiv griddots labelcolor labelsize* `Grid`

Coordinates should all be integers. Font needs to be defined before invoking this procedure. This could probably be simplified.

```
\pst@def{Grid}<%
  /a 4 string def            % Empty string
  /b ED                      % Label size
```

```
/d ED                       % Label color procedure.
/n ED                       % Number of grid dots
cvi dup 1 lt { pop 1 } if /c ED   % Number subdivisions
c div dup 0 eq { pop 1 } if /cy ED
c div dup 0 eq { pop 1 } if /cx ED  % division spacing
cy div cvi /y ED     % origin y
cx div cvi /x ED     % origin x
cy div cvi /y2 ED    % y2
cx div cvi /x2 ED    % x2
cy div cvi /y1 ED    % y1
cx div cvi /x1 ED    % x1
/h y2 y1 sub 0 gt { 1 } { -1 } ifelse def  % Sign of y2-y1
/w x2 x1 sub 0 gt { 1 } { -1 } ifelse def  % Sign of x2-x1
b 0 gt
{ /z1 b 4 div CLW 2 div add def
  /Helvetica findfont b scalefont setfont
  /b b .95 mul CLW 2 div add def }
if
gsave
n 0 gt
{ 1 setlinecap [ 0 cy n div ] 0 setdash }
{ 2 setlinecap }
ifelse
/c x1 def /i 500 w mul x1 add def        % Index
/e y cy mul def /f y1 cy mul def /g y2 cy mul def
x1 cx mul 0 T
{ newpath
  0 e moveto
  b 0 gt
  { gsave d c a cvs dup
    stringwidth pop /z2 ED
    w 0 gt {z1} {z1 z2 add neg} ifelse
    h 0 gt {b neg} {z1} ifelse
    rmoveto show grestore } if
  0 f moveto 0 g L stroke
  cx w mul 0 T
  c x2 eq c i eq or {exit} if
  /c c w add def
} loop
grestore
gsave
n 0 gt
{ 1 setlinecap [ 0 cx n div ] 0 setdash }
{ 2 setlinecap }
ifelse
/c y1 def /i 500 h mul y1 add def
/e x cx mul def /f x1 cx mul def /g x2 cx mul def
0 y1 cy mul T
{ newpath
  e 0 moveto
  b 0 gt { gsave d
    c a cvs dup
    stringwidth pop /z2 ED
    w 0 gt {z1 z2 add neg} {z1} ifelse
```

```
      h 0 gt {z1} {b neg} ifelse
      rmoveto show grestore } if
    f 0 moveto g 0 L stroke
    0 cy h mul T
    c y2 eq c i eq or {exit} if
    /c c h add def
  } loop
  grestore>
```

\psgrid

```
\def\psgrid{\def\pst@par{}\pst@object{psgrid}}
\def\psgrid@i{\@ifnextchar(%
  {\psgrid@ii}{\expandafter\psgrid@iv\pic@coor}}
\def\psgrid@ii(#1){\@ifnextchar(%
  {\psgrid@iii(#1)}{\psgrid@iv(0,0)(0,0)(#1)}}
\def\psgrid@iii(#1)(#2){\@ifnextchar(%
  {\psgrid@iv(#1)(#2)}{\psgrid@iv(#1)(#1)(#2)}}
\def\psgrid@iv(#1)(#2)(#3){%
  \begin@SpecialObj
  \pst@getcoor{#1}\pst@tempa
  \pst@getcoor{#2}\pst@tempb
  \pst@@getcoor{#3}%
  \ifnum\psk@subgriddiv>1
    \addto@pscode{gsave
    \psk@subgridwidth SLW \pst@usecolor\pssubgridcolor
    \pst@tempb \pst@coor \pst@tempa
    \pst@number\psxunit \pst@number\psyunit
    \psk@subgriddiv\space \psk@subgriddots\space
    {} 0 \tx@Grid grestore}%
  \fi
  \addto@pscode{gsave
    \psk@gridwidth SLW \pst@usecolor\psgridcolor
    \pst@tempb \pst@coor \pst@tempa
    \pst@number\psxunit \pst@number\psyunit
    1 \psk@griddots\space { \pst@usecolor\psgridlabelcolor }
    \psk@gridlabels \tx@Grid grestore}%
  \end@SpecialObj}
```

29 LR-box commands

\ifpsmathbox, \everypsbox

```
\newif\ifpsmathbox
\psmathboxtrue
\def\pst@mathflag{\z@}
\newtoks\everypsbox
```

\pst@makenotverbbox

```
\long\def\pst@makenotverbbox#1#2{%
  \edef\pst@mathflag{%
    \ifpsmathbox\ifmmode\ifinner 1\else 2\fi\else \z@\fi\else \z@\fi}%
  \setbox\pst@hbox=\hbox{%
    \ifcase\pst@mathflag\or$\m@th\textstyle\or$\m@th\displaystyle\fi
```

```
    {\the\everypsbox#2}%
    \ifnum\pst@mathflag>\z@$\fi}%
  #1}
```

\pst@makeverbbox

There is no way to do this such that with

```
\psframebox{\aftergroup\foo}
```

`\foo` does not end up outside the box. That is why this is not the default mode.

```
\def\pst@makeverbbox#1{%
  \def\pst@afterbox{#1}%
  \edef\pst@mathflag{%
    \ifpsmathbox\ifmmode\ifinner 1\else 2\fi\else \z@\fi\else \z@\fi}%
  \afterassignment\pst@beginbox
  \setbox\pst@hbox\hbox}
\def\pst@beginbox{%
  \ifcase\pst@mathflag\or$\m@th\or$\m@th\displaystyle\fi
  \bgroup\aftergroup\pst@endbox
  \the\everypsbox}
\def\pst@endbox{%
  \ifnum\pst@mathflag>\z@$\fi
  \egroup
  \pst@afterbox}
```

\psverbboxtrue, \psverbboxfalse

```
\def\pst@makebox{\pst@@makebox}
\def\psverbboxtrue{\def\pst@@makebox{\pst@makeverbbox}}
\def\psverbboxfalse{\def\pst@@makebox{\pst@makenotverbbox}}
\psverbboxfalse
```

\pst@longbox, \pst@makelongbox

There is no way to do this such that with

```
\psframebox{\aftergroup\foo}
```

`\foo` does not end up outside the box. That is why this is not the default mode.

```
\def\pst@longbox{%
  \def\pst@makebox{%
    \gdef\pst@makebox{\pst@@makebox}%
    \pst@makelongbox}}
\def\pst@makelongbox#1{%
  \def\pst@afterbox{#1}%
  \edef\pst@mathflag{%
    \ifpsmathbox\ifmmode\ifinner 1\else 2\fi\else \z@\fi\else \z@\fi}%
  \setbox\pst@hbox\hbox\bgroup
    \aftergroup\pst@afterbox
    \ifcase\pst@mathflag\or$\m@th\or$\m@th\displaystyle\fi
    \begingroup
      \the\everypsbox}
```

```
\def\pst@endlongbox{%
     \endgroup
     \ifnum\pst@mathflag>\z@$\fi
   \egroup}
```

\pslongbox

```
\def\pslongbox#1#2{%
  \@namedef{#1}{\pst@longbox#2}%
  \@namedef{end#1}{\pst@endlongbox}}
```

30 Frame boxes

framesep

```
\newdimen\psframesep
\def\psset@framesep#1{\pssetlength\psframesep{#1}}
\psset@framesep{3pt}
```

boxsep

```
\newif\ifpsboxsep
\def\psset@boxsep#1{\@nameuse{psboxsep#1}}
\psset@boxsep{true}
```

\pst@useboxpar

```
\def\pst@useboxpar{%
  \use@par
  \if@star
    \let\pslinecolor\psfillcolor
    \solid@star
    \let\solid@star\relax
  \fi
  \ifpsdoubleline \pst@setdoublesep \fi}
```

\psframebox

\psframebox puts its argument in an \hbox and draws a frame around it with thickness \pst@linewidth, and with distance \pst@framesep between each side of the frame (between the line making up each side) and each side of the box. The result is a box with no depth and with width and height equal to the width and height of the original box, plus 2(\pslinewidth+\psframesep).

\pst@dima is set to the distance between each side of the original box and the outer side of the frame (i.e., the side of the resulting box). \pst@dimb is set to the depth of the resulting box, \pst@dimc is set to the height plus depth of this box, and \pst@dimd is set to the width. \psframe does the drawing of the frame.

```
\def\psframebox{\def\pst@par{}\pst@object{psframebox}}
\def\psframebox@i{\pst@makebox\psframebox@ii}
\def\psframebox@ii{%
  \begingroup
    \pst@useboxpar
    \pst@dima=\pslinewidth
    \advance\pst@dima by \psframesep
```

```
    \pst@dimc=\wd\pst@hbox\advance\pst@dimc by \pst@dima
    \pst@dimb=\dp\pst@hbox\advance\pst@dimb by \pst@dima
    \pst@dimd=\ht\pst@hbox\advance\pst@dimd by \pst@dima
    \setbox\pst@hbox=\hbox{%
      \ifpsboxsep\kern\pst@dima\fi
      \begin@ClosedObj
        \addto@pscode{%
          \psk@cornersize
          \pst@number\pst@dima neg
          \pst@number\pst@dimb neg
          \pst@number\pst@dimc
          \pst@number\pst@dimd
          .5
          \tx@Frame}%
        \def\pst@linetype{2}%
        \showpointsfalse
      \end@ClosedObj
      \box\pst@hbox
      \ifpsboxsep\kern\pst@dima\fi}%
    \ifpsboxsep\dp\pst@hbox=\pst@dimb\ht\pst@hbox=\pst@dimd\fi
    \leavevmode\box\pst@hbox
  \endgroup}
```

\psdblframebox

```
\def\psdblframebox{\def\pst@par{}\pst@object{psdblframebox}}
\def\psdblframebox@i{\addto@par{doubleline=true}\psframebox@i}
```

\psclip, \endclip

Clipping involves drawing graphics objects, not grouped by `gsave` and `grestore`, which may affect the graphics environment. Furthermore, to reset the clipping path, we must either use `grestore` or `initclip`, neither of which is robust.

```
\def\psclip#1{%
  \leavevmode
  \begingroup
    \begin@psclip
    \begingroup
      \def\use@pscode{%
        \pstVerb{%
          \pst@dict
            /mtrxc CM def
            CP CP T
            \tx@STV
            \psk@origin
            \psk@swapaxes
            newpath
            \pst@code
            clip
            newpath
            mtrxc setmatrix
            moveto
            0 setgray
          end}%
```

```
        \gdef\pst@code{}}%
        \def\@multips(##1)(##2)##3##4{\pst@misplaced\multips}%
        \def\nc@object##1##2##3##4{\pst@misplaced{node connection}}%
        \hbox to\z@{#1}%
    \endgroup
  \def\endpsclip{%
    \end@psclip
    \endgroup}%
  \ignorespaces}
\def\endpsclip{\pst@misplaced\endpsclip}
\let\begin@psclip\relax
\def\end@psclip{\pstVerb{currentpoint initclip moveto}}
\def\AltClipMode{%
  \def\end@psclip{\pstVerb{\pst@grestore}}%
  \def\begin@psclip{\pstVerb{gsave}}}
```

\psclipbox

```
\def\clipbox{\@ifnextchar[{\psclipbox@}{psclipbox@[\z@]}}
\def\clipbox@[#1]{\pst@makebox\psclipbox@@{#1}}
\def\clipbox@@#1{%
  \pssetlength\pst@dimg{#1}%
  \leavevmode\hbox{%
    \begin@psclip
    \pst@Verb{%
      CM \tx@STV CP T newpath
      /a \pst@number\pst@dimg def
      /w \pst@number{\wd\pst@hbox}a add def
      /d \pst@number{\dp\pst@hbox}a add neg def
      /h \pst@number{\ht\pst@hbox}a add def
      a neg d moveto
      a neg h L
      w h L
      w d L
      closepath
      clip
      newpath
      0 0 moveto
      setmatrix}%
    \unhbox\pst@hbox
    \end@psclip}}
```

\psshadowbox

```
\def\psshadowbox{%
  \def\pst@par{}\pst@object{psshadowbox}}
\def\psshadowbox@i{\pst@makebox\psshadowbox@ii}
\def\psshadowbox@ii{%
  \begingroup
    \pst@useboxpar
    \psshadowtrue
    \psboxseptrue
    \def\psk@shadowangle{-45 }%
    \setbox\pst@hbox=\hbox{\psframebox@ii}%
    \pst@dimh=\psk@shadowsize\p@
```

```
    \pst@dimh=.7071\pst@dimh
    \pst@dimg=\dp\pst@hbox
    \advance\pst@dimg\pst@dimh
    \dp\pst@hbox=\pst@dimg
    \pst@dimg=\wd\pst@hbox
    \advance\pst@dimg\pst@dimh
    \wd\pst@hbox=\pst@dimg
    \leavevmode
    \box\pst@hbox
  \endgroup}
%
%
% \begin{macro}{\pscirclebox}
% "\pscirclebox@ii"'s argument is a hook that is used by node commands.
%    \begin{macrocode}
\def\pscirclebox{\def\pst@par{}\pst@object{pscirclebox}}
\def\pscirclebox@i{\pst@makebox{\pscirclebox@ii{}}}
\def\pscirclebox@ii#1{%
  \begingroup
    \pst@useboxpar
    \setbox\pst@hbox=\hbox{#1\pscirclebox@iii\box\pst@hbox}%
    \ifpsboxsep
      \pst@dima=.5\wd\pst@hbox
      \pst@pyth\pst@dima\pst@dimb\pst@dimc
      \advance\pst@dimc\pslinewidth
      \advance\pst@dimc\psframesep
      \setbox\pst@hbox=\hbox to2\pst@dimc{%
        \hss
        \vbox{\vskip\pst@dimc\vskip-\pst@dimb\box\pst@hbox}%
        \hss}%
      \advance\pst@dimc-\pst@dimb
      \dp\pst@hbox=\pst@dimc
    \fi
    \leavevmode\box\pst@hbox
  \endgroup}
\def\pscirclebox@iii{%
  \if@star
    \pslinewidth\z@
    \pstverb{\pst@dict \tx@STP \pst@usecolor\psfillcolor
      newpath \pscirclebox@iv \tx@SD end}%
  \else
    \begin@ClosedObj
    \def\pst@linetype{4}\showpointsfalse
    \addto@pscode{%
      \pscirclebox@iv CLW 2 div add 0 360 arc closepath}%
    \end@ClosedObj
  \fi}
\def\pscirclebox@iv{%
  \pst@number{\wd\pst@hbox}2 div
  \pst@number{\ht\pst@hbox}\pst@number{\dp\pst@hbox}add 2 div
  2 copy \pst@number{\dp\pst@hbox}sub 4 2 roll
  \tx@Pyth \pst@number\psframesep add }
```

\psovalbox

The argument of \psovalbox@ii is a hook used by node commands.

```
\def\psovalbox{\def\pst@par{}\pst@object{psovalbox}}
\def\psovalbox@i{\pst@makebox{\psovalbox@ii{}}}
\def\psovalbox@ii#1{%
  \begingroup
    \pst@useboxpar
    \pst@dimd=.707\pslinewidth\advance\pst@dimd by 1.414\psframesep
    \pst@dimg=\ht\pst@hbox\advance\pst@dimg\dp\pst@hbox
    \pst@dimb=.707\pst@dimg\advance\pst@dimb\pst@dimd
    \pst@dima=.707\wd\pst@hbox\advance\pst@dima\pst@dimd
    \setbox\pst@hbox=\hbox{#1\psovalbox@iii\box\pst@hbox}%
    \ifpsboxsep
      \setbox\pst@hbox\hbox to 2\pst@dima{\hss\unhbox\pst@hbox\hss}%
      \advance\pst@dimb-.5\pst@dimg
      \pst@dimg\ht\pst@hbox
      \advance\pst@dimg\pst@dimb
      \ht\pst@hbox=\pst@dimb
      \pst@dimg=\dp\pst@hbox
      \advance\pst@dimg\pst@dimb
      \dp\pst@hbox=\pst@dimb
    \fi
    \leavevmode\box\pst@hbox
  \endgroup}
\def\psovalbox@iii{%
  \begin@ClosedObj
  \addto@pscode{%
     0 360
     \pst@number\pst@dima \pst@number\pst@dimb
     \pst@number{\wd\pst@hbox}2 div
     \pst@number\pst@dimg 2 div \pst@number{\dp\pst@hbox}sub
     \tx@Ellipse
     closepath}%
  \def\pst@linetype{2}%
  \end@ClosedObj}
```

31 Circles, discs and ellipses

\psset@arcsep, \psk@arcsepA, \psk@arcsepB

```
\def\psset@arcsepA#1{\pst@getlength{#1}\psk@arcsepA}
\def\psset@arcsepB#1{\pst@getlength{#1}\psk@arcsepB}
\def\psset@arcsep#1{%
  \psset@arcsepA{#1}\let\psk@arcsepB\psk@arcsepA}
\psset@arcsep{0}
```

\tx@Arc

Syntax:

angle {*arrow*} {*add/sub*} ArcArrow *angle*

r=radius and c=57.2957/r should also be defined.

```
\pst@def{ArcArrow}<%
  /d ED     % add/sub
  /b ED     % arrow procedure
  /a ED     % angle
  gsave
    newpath
    0 -1000 moveto
    clip              % Set clippath far from arrow.
    newpath
    0 1 0 0 b         % Draw arrow to determine length.
  grestore
  c mul
  /e ED               % /e equals angle to adjust for arrow length.
  pop pop pop
  r a e d \tx@PtoC    % 'a e d' is end angle for arrow.
  y add exch x add exch
  r a \tx@PtoC        % Now arrow end coor and begin coor are on stack.
  y add exch x add exch
  b pop pop pop pop   % Draw arrow, and discard coordinates.
  a e d                   % End angle of arrow.
  CLW 8 div c mul neg d>  % Adjust angle to give a little overlap.
```

\psarc

```
\def\psarc{\def\pst@par{}\pst@object{psarc}}
\def\psarc@i{%
  \@ifnextchar({\psarc@iii}{\psarc@ii}}
\def\psarc@ii#1{\addto@par{arrows=#1}%
  \@ifnextchar({\psarc@iii}{\psarc@iii(0,0)}}
\def\psarc@iii(#1)#2#3#4{%
  \begin@OpenObj
    \pst@getangle{#3}\pst@tempa
    \pst@getangle{#4}\pst@tempb
    \pst@@getcoor{#1}%
    \pssetlength\pst@dima{#2}%
    \addto@pscode{\psarc@iv \psarc@v}%
    \gdef\psarc@type{0}%
    \showpointsfalse
  \end@OpenObj}
\def\psarc@iv{%
  \pst@coor /y ED /x ED
  /r \pst@number\pst@dima def
  /c 57.2957 r \tx@Div def
  /angleA
    \pst@tempa
    \psk@arcsepA c mul 2 div
    \ifcase \psarc@type add \or sub \fi
  def
  /angleB
    \pst@tempb
    \psk@arcsepB c mul 2 div
    \ifcase \psarc@type sub \or add \fi
  def
```

```
  \ifshowpoints\psarc@showpoints\fi
  \ifx\psk@arrowA\@empty
    \ifnum\psk@liftpen=2
      r angleA \tx@PtoC
      y add exch x add exch
      moveto
    \fi
  \fi}
\def\psarc@v{%
  x y r
  angleA
  \ifx\psk@arrowA\@empty\else
    { ArrowA CP }
    { \ifcase\psarc@type add \or sub \fi }
    \tx@ArcArrow
  \fi
  angleB
  \ifx\psk@arrowB\@empty\else
    { ArrowB }
    { \ifcase\psarc@type sub \or add \fi }
    \tx@ArcArrow
  \fi
  \ifcase\psarc@type arc \or arcn \fi}
\def\psarc@type{0}
\def\psarc@showpoints{%
  gsave
    newpath
    x y moveto
    x y r \pst@tempa \pst@tempb
    \ifcase\psarc@type arc \or arcn \fi
    closepath
    CLW 2 div SLW
    [ \psk@dash\space ] 0 setdash stroke
  grestore }
```

\psarcn

```
\def\psarcn{\def\pst@par{}\pst@object{psarcn}}
\def\psarcn@i{\def\psarc@type{1}\psarc@i}
```

\pscircle

```
\def\pscircle{\def\pst@par{}\pst@object{pscircle}}
\def\pscircle@i{\@ifnextchar({\pscircle@do}{\pscircle@do(0,0)}}
\def\pscircle@do(#1)#2{%
  \if@star
    {\use@par\qdisk(#1){#2}}%
  \else
    \begin@ClosedObj
      \pst@@getcoor{#1}%
      \pssetlength\pst@dimc{#2}%
      \def\pst@linetype{4}%
      \addto@pscode{%
        \pst@coor
        \pst@number\pst@dimc
```

```
        \psk@dimen CLW mul sub
        0 360 arc
        closepath}%
      \showpointsfalse
    \end@ClosedObj
  \fi
  \ignorespaces}
```

\qdisk

```
\def\qdisk(#1)#2{%
  \def\pst@par{}%
  \begin@SpecialObj
    \pst@@getcoor{#1}%
    \pssetlength\pst@dimg{#2}%
    \addto@pscode{\pst@coor \pst@number\pst@dimg \tx@SD}%
  \end@SpecialObj}
```

\pswedge

```
\def\pswedge{\def\pst@par{}\pst@object{pswedge}}
\def\pswedge@i{\@ifnextchar({\pswedge@ii}{\pswedge@ii(0,0)}}
\def\pswedge@ii(#1)#2#3#4{%
  \begin@ClosedObj
  \pssetlength\pst@dimc{#2}
  \pst@getangle{#3}\pst@tempa
  \pst@getangle{#4}\pst@tempb
  \pst@@getcoor{#1}%
  \def\pst@linetype{1}%
  \addto@pscode{%
    \pst@coor
    2 copy
    moveto
    \pst@number\pst@dimc \psk@dimen CLW mul sub % Adjusted radius
    \pst@tempa \pst@tempb
    arc
    closepath}%
    \showpointsfalse
  \end@ClosedObj}
```

Ellipse

Syntax:

angle1 angle2 x-radius y-radius x-origin y-origin `Ellipse`

```
\pst@def{Ellipse}<%
  /mtrx CM def
  T
  scale
  0 0 1 5 3 roll arc
  mtrx setmatrix>
```

\psellipse

```
\def\psellipse{\def\pst@par{}\pst@object{psellipse}}
\def\psellipse@i(#1){\@ifnextchar(%
  {\psellipse@ii(#1)}{\psellipse@ii(0,0)(#1)}}
\def\psellipse@ii(#1)(#2){%
  \begin@ClosedObj
    \pst@getcoor{#1}\pst@tempa
    \pst@@getcoor{#2}%
    \addto@pscode{%
      0 360
      \pst@coor
      \ifdim\psk@dimen\p@=\z@\else
        \psk@dimen CLW mul dup 3 1 roll
        sub 3 1 roll sub exch
      \fi
      \pst@tempa
      \tx@Ellipse
      closepath}%
    \def\pst@linetype{2}%
  \end@ClosedObj}
```

32 Repetition

\multirput

```
\def\multirput{%
  \begingroup\pst@getref{\pst@getrputrot\multirput@i}}
\def\multirput@i(#1){\@ifnextchar(%
  {\multirput@ii(#1)}{\multirput@ii(0,0)(#1)}}
\def\multirput@ii(#1,#2)(#3,#4)#5{%
  \pst@makebox{\multirput@iii(#1,#2)(#3,#4){#5}}}
\def\multirput@iii(#1,#2)(#3,#4)#5{%
  \pst@makesmall\pst@hbox
  \ifx\pst@rot\@empty\else\pst@rotate\pst@hbox\fi
  \pssetxlength\pst@dima{#1}\pssetylength\pst@dimb{#2}
  \pssetxlength\pst@dimc{#3}\pssetylength\pst@dimd{#4}
  \pst@cntg=#5\relax\pst@cnth=0\relax
  \leavevmode
  \loop\ifnum\pst@cntg>\pst@cnth
    \vbox to \z@{\vss\hbox to \z@{%
      \kern\pst@dima\copy\pst@hbox\hss}\vskip\pst@dimb}%
    \advance\pst@dima by\pst@dimc
    \advance\pst@dimb by\pst@dimd
    \advance\pst@cnth by 1
  \repeat
  \endgroup\ignorespaces}
```

\multips

```
\def\multips{\begingroup\pst@getrputrot\multips@i}
\def\multips@i(#1){\@ifnextchar({\@multips@ii(#1)}{\@multips@ii(0,0)(#1)}}
\def\@multips@ii(#1)(#2)#3#4{%
    \pst@getcoor{#1}\pst@tempa
    \pst@@getcoor{#2}%
    \pst@cnta=#3\relax
    \addto@pscode{%
```

```
      \pst@tempa T \the\pst@cnta\space \pslbrace
      gsave \ifx\pst@rot\@empty\else\pst@rot rotate \fi }%
    \hbox to\z@{%
      \def\init@pscode{%
        \addto@pscode{%
          gsave
          \pst@number\pslinewidth SLW
          \pst@usecolor\pslinecolor}}%
      \def\use@pscode{\addto@pscode{grestore}}%
      \def\psclip##1{\pst@misplaced\psclip}%
      \def\nc@object##1##2##3##4{\pst@misplaced{node connection}}%
      #4}%
    \addto@pscode{grestore \pst@coor T \psrbrace repeat}%
    \leavevmode
    \use@pscode
  \endgroup
  \ignorespaces}
```

33 Scaling

\scalebox

```
\def\scalebox#1{%
  \begingroup
    \pst@getscale{#1}\pst@tempa
    \pst@makebox{\@scalebox}}
\def\@scalebox{%
    \leavevmode
    \ifx\pst@tempa\@empty
      \box\pst@hbox
    \else
      \hbox{%
        \ht\pst@hbox=\pst@temph\ht\pst@hbox%
        \dp\pst@hbox=\pst@temph\dp\pst@hbox%
        \pst@dima=\pst@tempg\wd\pst@hbox%
        \ifdim\pst@dima<\z@\kern-\pst@dima\fi
        \pst@Verb{CP CP T \pst@tempa \tx@NET}%
        \hbox to \z@{\box\pst@hbox\hss}%
        \pst@Verb{%
          CP CP T
          1 \pst@tempg\space div 1 \pst@temph\space div scale
          \tx@NET}%
        \ifdim\pst@dima>\z@\kern\pst@dima\fi}%
    \fi
  \endgroup}
\pslongbox{Scalebox}{\scalebox}
```

\scaleboxto

```
\def\scaleboxto(#1,#2){%
  \begingroup
    \pssetlength\pst@dima{#1}%
    \pssetlength\pst@dimb{#2}%
    \pst@makebox{\@scaleboxto\@scalebox}}
\def\@scaleboxto{%
```

```
    \ifdim\pst@dima=\z@\else
      \pst@divide{\pst@dima}{\wd\pst@hbox}\pst@tempg
    \fi
    \ifdim\pst@dimb=\z@
      \let\pst@temph\pst@tempg
    \else
      \pst@dimc=\ht\pst@hbox\advance\pst@dimc\dp\pst@hbox
      \pst@divide{\pst@dimb}{\pst@dimc}\pst@temph
      \ifdim\pst@dima=\z@\let\pst@tempg\pst@temph\fi
    \fi
    \edef\pst@tempa{\pst@tempg\space\pst@temph\space scale }%
    \ifdim\pst@dima=\z@
      \ifdim\pst@dimb=\z@
        \@pstrickserr{%
          \string\scaleboxto\space dimensions cannot both be zero}\@ehpa
        \def\pst@tempa{}%
    \fi\fi}
\pslongbox{Scaleboxto}{\scaleboxto}
```

34 Rotation: The simple version

\tx@Rot

```
\pst@def{Rot}<\pstrotate>
```

\rotateleft, \rotateright, \rotatedown

These are pretty standard, except that they do not use `gsave` and `grestore`.

```
\def\rotateleft{\pst@makebox{\@rotateleft\pst@hbox}}
\def\@rotateleft#1{%
  \leavevmode\hbox{\hskip\ht#1\hskip\dp#1\vbox{\vskip\wd#1%
  \pst@Verb{90 \tx@Rot}
  \vbox to \z@{\vss\hbox to \z@{\box#1\hss}\vskip\z@}%
  \pst@Verb{-90 \tx@Rot}}}}
\def\rotateright{\pst@makebox{\@rotateright\pst@hbox}}
\def\@rotateright#1{%
  \hbox{\hskip\ht#1\hskip\dp#1\vbox{\vskip\wd#1%
  \pst@Verb{-90 \tx@Rot}
  \vbox to \z@{\hbox to \z@{\hss\box#1}\vss}%
  \pst@Verb{90 \tx@Rot}}}}
\def\rotatedown{\pst@makebox{\@rotatedown\pst@hbox}}
\def\@rotatedown#1{%
  \hbox{\hskip\wd#1\vbox{\vskip\ht#1\vskip\dp#1%
  \pst@Verb{180 \tx@Rot}%
  \vbox to \z@{\hbox to \z@{\box#1\hss}\vss}%
  \pst@Verb{-180 \tx@Rot}}}}
\pslongbox{Rotateleft}{\rotateleft}
\pslongbox{Rotateright}{\rotateright}
\pslongbox{Rotatedown}{\rotatedown}
```

35 \rput and company

`\rput` and similar commands are divided into fours steps:

1. The four arguments are collected:
   (a) The reference point argument is stored in \refpoint@x and \refpoint@y.
   (b) The rotation angle is store in \pst@rot.
   (c) The translation coordinate is passed to the command that is returned to after the box is made.
   (d) The RH-box is assigned to the register \pst@hbox.
2. The box is made zero-dimension and positioned at the reference point by \pst@makesmall.
3. The box is rotated by \pst@rotate.
4. The box is translated by \psput@.

35.1 Reference point

\pst@getref

```
\def\pst@getref#1{%
  \@ifnextchar[%
    {\def\refpoint@x{.5}\def\refpoint@y{.5}\pst@@getref{#1}}%
    {\let\refpoint@x\relax#1}}
\def\pst@@getref#1[#2]{%
  \pst@expandafter\pst@@@getref{#2}\@empty,,\@nil#1}
\def\pst@@@getref#1#2,#3,#4\@nil{%
  \ifx\@empty#3\@empty
    \@nameuse{getref@#1}\@nameuse{getref@#2}%
  \else
    \pst@checknum{#1#2}\refpoint@x
    \pst@checknum{#3}\refpoint@y
  \fi}
\def\getref@t{\def\refpoint@y{1}}
\def\getref@b{\def\refpoint@y{0}}
\def\getref@B{\let\refpoint@y\relax}
\def\getref@l{\def\refpoint@x{0}}
\def\getref@r{\def\refpoint@x{1}}
```

\pst@makesmall

```
\def\pst@makesmall#1{%
  \ifx\refpoint@x\relax
    \setbox#1=\hbox to\z@{\hss\vbox to \z@{\vss\box#1\vss}\hss}%
  \else
    \pst@@makesmall{#1}%
  \fi}
\def\pst@@makesmall#1{%
  \pst@dimh=\refpoint@x\wd#1%
  \ifx\refpoint@y\relax
    \pst@dimg=\dp#1%
  \else
    \pst@dimg=\refpoint@y\ht#1%
    \advance\pst@dimg\refpoint@y\dp#1%
  \fi
  \setbox#1=\hbox to\z@{%
```

```
    \hskip-\pst@dimh\vbox to\z@{\vss\box#1\vskip-\pst@dimg}\hss}}
```

35.2 Rotation

\pst@getrputrot

```
\def\pst@getrputrot#1{%
  \@ifnextchar(%
    {\def\pst@rot{}#1}%
    {\pst@getrot{\@ifnextchar({#1}{#1(0,0)}}}}
```

\pst@getrot

```
\def\pst@getrot#1#2{%
  \pst@expandafter{\@ifnextchar*{\pst@@@getrot}{\pst@@getrot}}{#2}\@nil
  \ifx\pst@rotlist\@empty\else
    \edef\pst@rotlist{\pst@rotlist \pst@rot add }%
  \fi
  #1}
\def\pst@@getrot#1\@nil{%
  \def\next##1@#1=##2@##3\@nil{%
    \ifx\relax##2%
      \pst@getangle{#1}\pst@rot
    \else
      \def\pst@rot{##2}%
    \fi}%
  \expandafter\next\pst@rottable @#1=\relax @\@nil}
\def\pst@@@getrot#1#2\@nil{%
  \pst@@getrot#2\@nil
  \edef\pst@rot{\pst@rotlist neg \ifx\pst@rot\@empty\else\pst@rot add \fi}}%
\def\pst@rotlist{0 }
\def\pst@rot{}
```

\pst@rottable

The trailing spaces must be included, except when empty.

```
\def\pst@rottable{%
  @0=%
  @U=%
  @L=90 %
  @D=180 %
  @R=-90 %
  @N=\pst@rotlist neg %
  @W=\pst@rotlist neg 90 add %
  @S=\pst@rotlist neg 180 add %
  @E=\pst@rotlist neg 90 sub }
```

\pst@rotate

The last argument should be the register for a zero-dimensional box that is to be rotated. By first putting the box in a zero-dimension box centered at the reference point of the original box, we do not have to use `gsave` and `grestore`.

```
\def\pst@rotate#1{%
  \setbox#1=\hbox{%
```

```
    \pst@Verb{\pst@rot \tx@Rot}%
    \box#1%
    \pst@Verb{\pst@rot neg \tx@Rot}}}
```

35.3 Translation

\psput@cartesian, \psput@special

\psput@ is defined by the \NormalCoor and \SpecialCoor commands to invoke either \psput@cartesian or \psput@special.

\psput@cartesian is for Cartesian coordinates only. TEX does the translation.

\psput@special works for any coordinates. PostScript does the translation. /lmtrx is used to store a stack of transformation for nested translations.

```
\def\psput@cartesian#1{%
  \hbox to \z@{\kern\pst@dimg{\vbox to \z@{\vss\box#1\vskip\pst@dimh}\hss}}}
\def\psput@special#1{%
  \hbox{%
    \pst@Verb{{ \pst@coor } \tx@PutCoor \tx@PutBegin}%
    \box#1%
    \pst@Verb{\tx@PutEnd}}}
\pst@def{PutCoor}<%
  gsave
    CP T
    CM
    \tx@STV
    exch exec
    moveto
    setmatrix
    CP
  grestore>
\pst@def{PutBegin}<%
  /lmtrx [ tx@Dict /lmtrx known { lmtrx aload pop } if CM ] def
  CP 4 2 roll T moveto>
\pst@def{PutEnd}<CP /lmtrx [ lmtrx aload pop setmatrix ] def moveto>
```

35.4 The real thing

\begin@psput, \end@psput

```
\def\begin@psput#1{\begingroup\pst@killglue\leavevmode\pst@ifstar{#1}}%
\def\end@psput#1(#2){%
  \pst@makebox{%
    \if@star
      \setbox\pst@hbox\hbox{\psframebox*[boxsep=false]{\unhbox\pst@hbox}}%
    \fi
    #1(#2)%
    \endgroup
    \ignorespaces}}
```

\rput

```
\def\rput{\begin@psput{\pst@getref{\pst@getrputrot{\end@psput\rput@i}}}}
\def\rput@i(#1){%
```

```
    \pst@makesmall\pst@hbox
    \ifx\pst@rot\@empty\else\pst@rotate\pst@hbox\fi
    \psput@{#1}\pst@hbox}
```

\cput

The first argument of \cput@iii is a hook used by node commands.

```
\def\cput{\def\pst@par{}\pst@object{cput}}
\def\cput@i{\begingroup\pst@killglue\leavevmode\pst@getrputrot\cput@ii}
\def\cput@ii(#1){\pst@makebox{\cput@iii{}(#1)}}
\def\cput@iii#1(#2){%
    \setbox\pst@hbox=\hbox{\psboxsepfalse\pscirclebox@ii{#1}}%
    \let\refpoint@x\relax
    \rput@i(#2)%
  \endgroup
  \ignorespaces}
```

36 \uput and company

The difference between \uput and \rput is that \rput's reference point is replaced by labelsep and reference angle arguments.

\psset@labelsep, \pslabelsep

```
\newdimen\pslabelsep
\def\psset@labelsep#1{\pssetlength\pslabelsep{#1}}
\psset@labelsep{5pt}
```

\pst@getrefangle

```
\def\pst@getrefangle#1\@nil{%
  \def\next##1@#1=##2"##3@##4\@nil{%
    \ifx\relax##2%
      \pst@getangle{#1}\pst@refangle
      \def\pst@uputref{}%
    \else
      \edef\pst@refangle{##2}%
      \edef\pst@uputref{##3}%
    \fi}%
  \expandafter\next\pst@refangletable @#1=\relax"@\@nil}
```

\pst@refangletable

```
\def\pst@refangletable{%
  @r=0"20%
  @u=90"02%
  @l=180"10%
  @d=-90"01%
  @ur=45"22%
  @ul=135"12%
  @dr=-135"21%
  @dl=-45"11}
```

\uput

```
\def\uput{\begin@psput{\@ifnextchar[{\uput@ii}{\uput@i}}}
\def\uput@i#1{\pssetlength\pslabelsep{#1}\uput@ii}
\def\uput@ii[#1]{%
  \pst@expandafter\pst@getrefangle{#1}\@nil
  \pst@getrputrot{\end@psput\uput@iii}}
\def\uput@iii(#1){%
    \ifx\pst@uputref\@empty
      \uput@iv\tx@UUput
    \else
      \ifx\pst@rot\@empty
        \expandafter\uput@v\pst@uputref
      \else
        \uput@iv\tx@UUput
      \fi
    \fi
    \psput@{#1}\pst@hbox}
\def\uput@iv#1{%
  \edef\pst@coor{%
    \pst@number\pslabelsep
    \pst@number{\wd\pst@hbox}%
    \pst@number{\ht\pst@hbox}%
    \pst@number{\dp\pst@hbox}%
    \pst@refangle\space \ifx\pst@rot\@empty\else\pst@rot\space sub \fi
    \tx@Uput #1}%
  \setbox\pst@hbox=\hbox to\z@{\hss\vbox to\z@{\vss\box\pst@hbox\vss}\hss}%
  \setbox\pst@hbox=\psput@special\pst@hbox
  \ifx\pst@rot\@empty\else\pst@rotate\pst@hbox\fi}
\def\uput@v#1#2{%
  \ifnum#1>\z@\ifnum#2>\z@\pslabelsep=.707\pslabelsep\fi\fi
  \setbox\pst@hbox=\vbox to\z@{%
    \ifnum#2=1 \vskip\pslabelsep\else\vss\fi
    \hbox to\z@{%
      \ifnum#1=2 \hskip\pslabelsep\else\hss\fi
      \box\pst@hbox
      \ifnum#1=1 \hskip\pslabelsep\else\hss\fi}%
    \ifnum#2=2 \vskip\pslabelsep\else\vss\fi}}
```

\tx@Uput

I forget how this works, but it does.

```
\pst@def{Uput}<%
  /a ED
  add 2 div /h ED
  2 div /w ED
  /s a sin def
  /c a cos def
  /b
  s abs c abs 2 copy gt
  dup /q ED
  { pop } { exch pop } ifelse
  def
  /w1 c b div w mul def
  /h1 s b div h mul def
  q
```

```
  { w1 abs w sub dup c mul abs }
  { h1 abs h sub dup s mul abs }
  ifelse>
\pst@def{UUput}<%
  /z ED
  abs /y ED
  /x ED
  q
  { x s div c mul abs y gt }
  { x c div s mul abs y gt }
  ifelse
  { x x mul y y mul sub z z mul add sqrt z add }
  { q { x s div } { x c div } ifelse abs }
  ifelse
  a \tx@PtoC h1 add exch w1 add exch>
```

\pst@getlabelsep, \Rput

\Rput is an obsolete version of \uput.

```
\def\pst@getlabelsep#1{%
  \@ifnextchar[%
    {\def\refpoint@x{.5}\def\refpoint@y{.5}\pst@@getref{#1}}%
    {\pst@@getlabelsep{#1}}}
\def\pst@@getlabelsep#1#2{\pssetlength\pslabelsep{#2}\pst@getref{#1}}
\def\Rput{%
  \begin@psput{\pst@getlabelsep{\pst@getrputrot{\end@psput{\Rput@i\rput@i}}}}}
\def\Rput@i{%
  \pst@dimg=\dp\pst@hbox
  \advance\pst@dimg\pslabelsep
  \dp\pst@hbox=\pst@dimg
  \pst@dimg=\ht\pst@hbox
  \advance\pst@dimg\pslabelsep
  \ht\pst@hbox=\pst@dimg
  \setbox\pst@hbox\hbox{\kern\pslabelsep\box\pst@hbox\kern\pslabelsep}}%
```

37 Pictures

\pspicture

```
\def\pspicture{\begingroup\pst@ifstar\pst@picture}
\def\pst@picture{%
  \@ifnextchar[{\pst@@picture}{\pst@@picture[0]}}
\def\pst@@picture[#1]#2(#3,#4){%
 \@ifnextchar({\pst@@@picture[#1](#3,#4)}%
   {\pst@@@picture[#1](0,0)(#3,#4)}}
\def\pst@@@picture[#1](#2,#3)(#4,#5){%
  \pssetxlength\pst@dima{#2}\pssetylength\pst@dimb{#3}%
  \pssetxlength\pst@dimc{#4}\pssetylength\pst@dimd{#5}%
  \def\pst@tempa{#1}%
  \setbox\pst@hbox=\hbox\bgroup
  \begingroup\KillGlue
  \@ifundefined{@latexerr}{}{\let\unitlength\psunit}%
```

```
  \edef\pic@coor{(#2,#3)(#2,#3)(#4,#5)}\ignorespaces}
\def\pic@coor{(0,0)(0,0)(10,10)}
\def\endpspicture{%
    \pst@killglue
    \endgroup
    \egroup
    \ifdim\wd\pst@hbox=\z@\else
      \@pstrickserr{Extraneous space in the pspicture environment}%
        {Type \space <return> \space to procede.}%
    \fi
    \ht\pst@hbox=\pst@dimd
    \dp\pst@hbox=-\pst@dimb
    \setbox\pst@hbox=\hbox{%
      \kern-\pst@dima
      \ifx\pst@tempa\@empty\else
        \advance\pst@dimd-\pst@dimb
        \pst@dimd=\pst@tempa\pst@dimd
        \advance\pst@dimd\pst@dimb
        \lower\pst@dimd
      \fi
      \box\pst@hbox
      \kern\pst@dimc}%
    \if@star\setbox\pst@hbox=\hbox{\clipbox@@\z@}\fi
    \leavevmode\box\pst@hbox
  \endgroup}
\@namedef{pspicture*}{\pspicture*}
\@namedef{endpspicture*}{\endpspicture}
```

38 Overlays

Overlays work by translating invisible material. They take advantage of the fact that PostScript is running parallel to TEX, and so we can redefine the value of some PostScript variables in order to get a different overlay printed each time we output a box containing overlay commands (even though the box has already been typeset by TEX).

BeginOverlay

`BeginOL` is a PostScript procedure, with syntax:

(*string*) `BeginOL`

If the string is not `(all)` and does not match `TheOL`, then the output is made invisible by translating it over by the coffee pot (actually, by a distance `OLUnit`). Otherwise, it is made visible by translating it back to the page.

Rather than translating the page, we could define a small clipping path off the page, but that would be more likely to be messed up by someone's `initclip` (e.g., by PSTricks' `initclip`!).

```
\pst@def{BeginOL}<%
  dup (all) eq exch TheOL eq or
  { IfVisible not
    { CP OLUnit T moveto
      /IfVisible true def }
```

```
    if }
  { IfVisible
    { CP OLUnit \tx@NET moveto
      /IfVisible false def }
    if }
  ifelse>
```

InitOL

This figures out how far in the current units used by the driver is 50 inches up and to the right. This works even though drivers use unusual coordinate systems (even `dvips`). This macro also defines `BOL` to be `BeginOL` and sets the default value of `IfVisible`.

```
\pst@dimg=40in
\edef\pst@OLunit{\pst@number\pst@dimg}
\pst@def{InitOL}<%
  /OLUnit [ gsave CM \tx@STV \pst@OLunit
    dup moveto setmatrix CP grestore ] cvx def
  /BOL { \tx@BeginOL } def /IfVisible true def>
```

\pst@initoverlay

This defines `TheOL` to be `#1`. It must be inserted just before printing overlay `#1`.

```
\def\pst@initoverlay#1{\pst@Verb{\tx@InitOL /TheOL (#1) def}}
```

\pst@overlay, \pst@endoverlay

`\pst@overlay` just calls `BeginOverlay`.

```
\def\pst@overlay#1{%
  \edef\curr@overlay{#1}%
  \pst@Verb{(#1) BOL}%
  \aftergroup\pst@endoverlay}
\def\pst@endoverlay{%
  \pst@Verb{(\curr@overlay) BOL}}
\def\curr@overlay{all}
```

\overlaybox, \endoverlaybox, \putoverlaybox

`\pst@initoverlay`, `\pst@overlay`, and `\pst@endoverlay` are the overlays primitives. An interface must be set up that guarantees that `\pst@overlay` and `\pst@endoverlay` are only used inside a box, and that `\pst@initoverlay` is inserted each type the box is printed. Here is one such interface (see `seminar.sty` for an interface for slides). The extra `\begingroup` and `\endgroup` assure that each `\pst@endoverlay` is executed within the box.

```
\newbox\theoverlaybox
\def\overlaybox{%
  \setbox\theoverlaybox=\hbox\bgroup
    \begingroup
    \let\psoverlay\pst@overlay
    \def\overlaybox{%
      \@pstrickserr{Overlays cannot be nested}\@eha}%
    \def\putoverlaybox{%
```

```
      \@pstrickserr{You must end the overlay box
      before using \string\putoverlaybox}}%
    \psoverlay{main}}
\def\endoverlaybox{\endgroup\egroup}
\def\putoverlaybox#1{%
  \hbox{\pst@initoverlay{#1}\copy\theoverlaybox}}
\def\psoverlay{\@pstrickserr{\string\psoverlay\space
  can only be used after \string\overlaybox}}
```

39 Configuration file – revisited

```
\ifx\pstcustomize\relax \input pstricks.con \fi
\pst@ATH<end>
\catcode`\@=\PstAtCode\relax
\endinput
```

Part I

pst-node.doc

Check whether file has been loaded already.

```
\csname PSTnodesLoaded\endcsname
\let\PSTnodesLoaded\endinput
```

Load `pstricks.tex` if necessary:

```
\ifx\PSTricksLoaded\endinput\else
  \def\next{\input pstricks.tex}\expandafter\next
\fi
```

Take care of the catcode of `@`:

```
\edef\TheAtCode{\the\catcode`\@}
\catcode`\@=11
```

40 Node header

Nodes use the dictionary `tx@NodeDict`, which is always put on the stack after `tx@Dict`. `tx@NodeDict` should avoid using the same procedure names as are found in `tx@Dict`, especially those that do not use scratch variables and hence can be used without problem when `tx@NodeDict` is on top of the stack. When invoking a `tx@Dict` procedure that does use scratch variables, `tx@Dict` should be put on top.

\pst@nodedict

```
\pst@ATH<\% Version \fileversion, \filedate.>
\pst@ATH<\% For use with \pstdriver.>
\pst@ATH</tx@NodeDict 200 dict def tx@NodeDict begin>
\ifx\pst@useheader\iftrue
  \pstheader{pst-node.pro}
  \def\pst@nodedict{tx@NodeDict begin }
\else
  \def\pst@nodedict{%
    /tx@NodeDict where
    { pop }
    { userdict begin /tx@NodeDict 200 dict def end }
    ifelse
    tx@NodeDict begin }
\fi
```

41 Nodes

\pst@getnode

```
% A node is a dictionary. To reduce the chance of errors, we check that the
% name begins with a letter and does not contain any spaces.
%    \begin{macrocode}
\def\pst@getnode#1#2{%
  \pst@expandafter\pst@@getnode{#1} * \@nil{#1}#2}
```

```
\def\pst@@getnode#1#2 #3\@nil#4#5{%
  \ifcat#1a\relax
    \def#5{/TheNode#1#2 }%
  \else
    \def#5{/BadNode }%
    \@pstrickserr{Bad node name: `#4'}\@ehpa
  \fi}
```

Before a node is defined, the coordinate system is scaled to PSTricks' standard coordinate system, with the origin at TEX's current point. The following objects should then be added to the node dictionary:

NodeMtrx The current matrix.

X The x-coordinate of the center.

Y The y-coordinate of the center.

NodePos A procedure that, given the values of `Sin`, `Cos`, and `Nodesep`, gives the relative position of the point that is distance `Nodesep`from the edge of the node, in the direction `(Cos,Sin)` from the center. "Relative" means relative to `(X,Y)` and for the coordinate system in effect when the node was defined.

\tx@NewNode, \pst@newnode

The node's dictionary size should be large enough for the 7 key's mentioned above, plus any keys the node needs for `NodePos`, plus a few more to avoid mistakes.

Syntax for `NewNode`

{*beforenode˙proc*} /*node˙name dict˙size* {*node˙proc*} NewNode"

`<beforenode_proc>` is stuff to be done with `tx@Dict`. It might leave things on the stack for use by `<node_proc>`.

Syntax for `\pst@newnode`:

\pst@newnode{*node˙name*}{*dict˙size*}{*beforenode˙proc*}{*node˙proc*}

```
\pst@def{NewNode}<%
  gsave
    /next ED
    dict
    dup 3 -1 roll ED
    begin
      tx@Dict begin
        \tx@STV
        CP T
        exec
      end
      /NodeMtrx CM def
      next
    end
  grestore>
\def\pst@newnode#1#2#3#4{%
  \leavevmode
```

```
  \pst@getnode{#1}\pst@thenode
  \pst@Verb{%
    \pst@nodedict
       { #3 } \pst@thenode #2 { #4 } \tx@NewNode
    end}}
```

\tx@InitPnode, \pnode

```
\pst@def{InitPnode}<%
  /Y ED /X ED
  /NodePos { Nodesep Cos mul Nodesep Sin mul } def>
\def\pnode{\@ifnextchar({\pnode@}{\pnode@(0,0)}}
\def\pnode@(#1)#2{%
  \pst@@getcoor{#1}%
  \pst@newnode{#2}{10}{\pst@coor}{\tx@InitPnode}%
  \ignorespaces}
```

\tx@InitCnode, \cnode

```
\pst@def{InitCnode}<%
  /r ED /Y ED /X ED
  /NodePos { Nodesep r add dup Cos mul exch Sin mul } def>
\def\cnode{\def\pst@par{}\pst@object{cnode}}
\def\cnode@i{\@ifnextchar({\cnode@ii}{\cnode@ii(0,0)}}
\def\cnode@ii(#1)#2#3{%
  \begingroup
    \use@par
    \pscircle@do(#1){#2}%
    \pst@@getcoor{#1}%
    \pssetlength\pst@dimc{#2}%
    \pst@newnode{#3}{11}{%
      \pst@coor
      \pst@number\pst@dimc
      \pst@number\pslinewidth
      \psk@dimen .5 sub mul sub}%
      {\tx@InitCnode}%
  \endgroup
  \ignorespaces}
\def\cnodeput{\def\pst@par{}\pst@object{cnodeput}}
\def\cnodeput@i{%
  \begingroup
    \pst@killglue
    \leavevmode
    \pst@getrputrot
    \cnodeput@ii}
\def\cnodeput@ii(#1)#2{%
  \pst@makebox{\cput@iii{\cnodeput@iii{#2}}(#1)}}
\def\cnodeput@iii#1{%
  \pst@newnode{#1}{11}{\pscirclebox@iv \pst@number\pslinewidth add}%
    {\tx@InitCnode}}
```

\circlenode

```
\def\circlenode{\def\pst@par{}\pst@object{circlenode}}
\def\circlenode@i#1{\pst@makebox{\pscirclebox@ii{\cnodeput@iii{#1}}}}
```

```
\pst@def{GetRnodePos}<%
  Cos 0 gt
  { /dx r Nodesep add def }
  { /dx l Nodesep sub def }
  ifelse
  Sin 0 gt
  { /dy u Nodesep add def }
  { /dy d Nodesep sub def }
  ifelse
  dx Sin mul abs dy Cos mul abs gt
  { dy Cos mul Sin div dy }
  { dx dup Sin mul Cos \tx@Div }
  ifelse>
```

InitRnode

Syntax:

yref ht dp bool xref wd InitRnode

Additional keys: `r`, `l`, `d`, `u`, `dx` and `dy`.

```
\pst@def{InitRnode}<%
  /r ED r mul neg /l ED /r r l add def
  /X l neg def
  { neg /d ED /u ED /Y 0 def }
  { neg /Y ED
    Y sub /u ED
    u mul neg /d ED
    /u u d add def
    /Y Y d sub def }
  ifelse
  /NodePos { \tx@GetRnodePos } def>
```

\rnode

The ability to set the refpoint is an undocumented feature that may be omitted.

```
\def\rnode{\begingroup\pst@getref\rnode@}
\def\rnode@#1{\pst@makebox{\rnode@@{#1}}}
\def\rnode@@#1{%
  \ifx\refpoint@x\relax
    \def\refpoint@y{.5}%
    \def\refpoint@x{.5}%
  \fi
  \pst@newnode{#1}{16}{}{%
    \ifx\refpoint@x\relax .5 \else \refpoint@y\space \fi
    \pst@number{\ht\pst@hbox}%
    \pst@number{\dp\pst@hbox}%
    \ifx\refpoint@y\@empty true \else false \fi
    \refpoint@x\space
    \pst@number{\wd\pst@hbox}%
    \tx@InitRnode}%
  \box\pst@hbox
  \endgroup}
```

InitRNode

Syntax:

ht dp wd xref yref `InitRNode`

```
\pst@def{InitRNode}<%
  /Y ED /X ED /r ED /X r 2 div X add def /r r X sub def /l X neg def
  Y add neg /d ED Y sub /u ED
  /NodePos { \tx@GetRnodePos } def>
```

\Rnode

```
\def\Rnode{\@ifnextchar({\Rnode@}{\Rnode@(\RnodeRef)}}
\def\Rnode@(#1)#2{\pst@makebox{\Rnode@@(#1){#2}}}
\def\Rnode@@(#1)#2{%
  \begingroup
    \pst@@getcoor{#1}%
    \pst@newnode{#2}{16}{%
      \pst@number{\ht\pst@hbox}\pst@number{\dp\pst@hbox}%
      \pst@number{\wd\pst@hbox}\pst@coor}{\tx@InitRNode}%
    \box\pst@hbox
  \endgroup}
\def\RnodeRef{0,.7ex}
```

GetOnodePos

```
\pst@def{GetOnodePos}<%
  /ww w Nodesep add def /hh h Nodesep add def
  Sin ww mul Cos hh mul \tx@Atan dup
  cos ww mul exch sin hh mul>
```

\ovalnode

Additional keys: `w`, `h`, `ww`, `hh`.

```
\def\ovalnode{\def\pst@par{}\pst@object{ovalnode}}
\def\ovalnode@i#1{\pst@makebox{\psovalbox@ii{\ovalnode@ii{#1}}}}
\def\ovalnode@ii#1{%
  \pst@newnode{#1}{14}{}{%
    /X \pst@number{\wd\pst@hbox}2 div def
    /Y \pst@number\pst@dimg 2 div \pst@number{\dp\pst@hbox}sub def
    /w \pst@number\pst@dima def
    /h \pst@number\pst@dimb def
    /NodePos { \tx@GetOnodePos } def}}
```

42 Node connections: Preliminaries

\tx@GetCenter, \tx@GetAngle

Syntax:

- `GetCenter` *x y* (Center coordinates)
- `GetAngle` *angle* (Angle from A to B)

```
\pst@def{GetCenter}<begin X Y NodeMtrx transform CM itransform end>
\pst@def{GetAngle}<%
  nodeA \tx@GetCenter
  nodeB \tx@GetCenter
  3 -1 roll sub 3 1 roll sub neg \tx@Atan>
```

\tx@GetEdge, \tx@GetPos

Syntax:

offset angle nodesep node `GetEdge` *x y*

`GetPos` defines `(x1,y1)` and `(x2,y2)` to be coordinates of position for node A and B, taking into account `AngleA`, `AngleB`, `OffsetA`, `OffsetB`, `NodesepA` and `NodesepB`.

```
\pst@def{GetEdge}<%
  begin
    /Nodesep ED
    dup
    1 0 NodeMtrx dtransform CM idtransform exch atan sub
    dup sin /Sin ED cos /Cos ED
    NodePos Y add exch X add exch
    NodeMtrx transform CM itransform
  end % offset angle x y
  4 2 roll
% Now add the offsets:
  1 index 0 eq
  { pop pop }
  { 2 copy 5 2 roll % x offset angle y offset angle
    cos mul add
    4 1 roll
    sin mul sub
    exch }
  ifelse>
\pst@def{GetPos}<%
  OffsetA AngleA NodesepA nodeA \tx@GetEdge /y1 ED /x1 ED
  OffsetB AngleB NodesepB nodeB \tx@GetEdge /y2 ED /x2 ED>

\def\check@arrow#1#2{%
  \check@@arrow#2-\@nil
  \if@pst
    \addto@par{arrows=#2}%
    \def\next{#1}%
  \else
    \def\next{#1{#2}}%
  \fi
  \next}
\def\check@@arrow#1-#2\@nil{%
  \ifx\@nil#2\@nil\@pstfalse\else\@psttrue\fi}

\pst@def{InitNC}<%
  /nodeB ED /nodeA ED
  /NodesepB ED /NodesepA ED
  /OffsetB ED /OffsetA ED
  tx@NodeDict nodeA known tx@NodeDict nodeB known and dup
  { /nodeA nodeA load def /nodeB nodeB load def } if>
```

```
\def\nc@object#1#2#3#4{%
  \begin@OpenObj
    \showpointsfalse
    \pst@getnode{#1}\pst@tempa
    \pst@getnode{#2}\pst@tempb
    \gdef\lputpos@default{#3}%
    \addto@pscode{%
      \pst@nodedict
        \psk@offsetA
        \psk@offsetB neg
        \psk@nodesepA
        \psk@nodesepB
        \pst@tempa
        \pst@tempb
        \tx@InitNC { #4 } if
      end}%
    \def\use@pscode{%
      \pst@Verb{gsave \tx@STV newpath \pst@code\space grestore}%
      \gdef\pst@code{}}%
  \end@OpenObj}
\def\lputpos@default{.5}

\def\pc@object#1{%
  \@ifnextchar({\pc@@object#1}{\pst@getarrows{\pc@@object#1}}}
\def\pc@@object#1(#2)(#3){%
  \pnode(#2){@@A}\pnode(#3){@@B}%
  #1{@@A}{@@B}}

\def\psset@nodesepA#1{\pst@getlength{#1}\psk@nodesepA}
\def\psset@nodesepB#1{\pst@getlength{#1}\psk@nodesepB}
\def\psset@nodesep#1{%
  \psset@nodesepA{#1}\let\psk@nodesepB\psk@nodesepA}
\psset@nodesep{0}
```

\psset@offset, \psk@offsetA, \psk@offsetB

```
\def\psset@offsetA#1{\pst@getlength{#1}\psk@offsetA}
\def\psset@offsetB#1{\pst@getlength{#1}\psk@offsetA}
\def\psset@offset#1{%
  \psset@offsetA{#1}\let\psk@offsetB\psk@offsetA}
\psset@offset{0}
```

\psset@arm, \psk@armA, \psk@armB

```
\def\psset@armA#1{\pst@getlength{#1}\psk@armA}
\def\psset@armB#1{\pst@getlength{#1}\psk@armB}
\def\psset@arm#1{\psset@armA{#1}\let\psk@armB\psk@armA}
\psset@arm{10pt}
```

\psset@angle, \psk@angleA, \psk@angleB

```
\def\psset@angleA#1{\pst@getangle{#1}\psk@angleA}
\def\psset@angleB#1{\pst@getangle{#1}\psk@angleB}%
\def\psset@angle#1{\pst@getangle{#1}\psk@angleA
  \let\psk@angleB\psk@angleA}
```

```
\psset@angle{0}
```

\psset@arcangle, \psk@arcangleA, \psk@arcangleB

```
\def\psset@arcangleA#1{\pst@getangle{#1}\psk@arcangleA}
\def\psset@arcangleB#1{\pst@getangle{#1}\psk@arcangleB}%
\def\psset@arcangle#1{\pst@getangle{#1}\psk@arcangleA
  \let\psk@arcangleB\psk@arcangleA}
\psset@arcangle{8}
```

\psset@ncurv, \psk@ncurvA, \psk@ncurvB

```
\def\psset@ncurvA#1{\pst@checknum{#1}\psk@ncurvA}
\def\psset@ncurvB#1{\pst@checknum{#1}\psk@ncurvB}%
\def\psset@ncurv#1{\psset@ncurvA{#1}\let\psk@ncurvB\psk@ncurvA}
\psset@ncurv{.67}
```

43 Node connections: The real thing

```
\pst@def{LineMP}<%
  4 copy
  1 t sub mul exch t mul add 3 1 roll
  1 t sub mul exch t mul add exch 6 2 roll
  sub 3 1 roll sub \tx@Atan>
```

\tx@NCCoor, \tx@NCLine

Syntax:

OffsetB NodesepB OffsetA NodesepA NCLine

Leaves coordinates on stack rather than actually drawing line.

```
\pst@def{NCCoor}<%
  \tx@GetAngle
  /AngleA ED /AngleB AngleA 180 add def
  \tx@GetPos
  /LPutVar [ x2 x1 y2 y1 ] cvx def
  /LPutPos { LPutVar \tx@LineMP } def
  x1 y1 x2 y2>
\pst@def{NCLine}<%
  \tx@NCCoor
  tx@Dict begin
    ArrowB
    4 2 roll
    ArrowA
    lineto
  end>

\def\ncline{\def\pst@par{}\pst@object{ncline}}
\def\ncline@i{\check@arrow{\ncline@ii}}
\def\ncline@ii#1#2{\nc@object{#1}{#2}{.5}{\tx@NCLine}}

\def\pcline{\def\pst@par{}\pst@object{pcline}}
\def\pcline@i{\pc@object\ncline@ii}
```

```
\def\ncLine{\def\pst@par{}\pst@object{ncLine}}
\def\ncLine@i{\check@arrow{\ncLine@ii}}
\def\ncLine@ii#1#2{\nc@object{#1}{#2}{.5}%
  {\tx@NCLine
  /LPutVar [
    nodeA \tx@GetCenter
    nodeB \tx@GetCenter
    3 1 roll 4 1 roll
  ] cvx def }}

\pst@def{BezierMidpoint}<%
  /y3 ED /x3 ED
  /y2 ED /x2 ED
  /y1 ED /x1 ED
  /y0 ED /x0 ED
  /t ED
  /cx x1 x0 sub 3 mul def
  /cy y1 y0 sub 3 mul def
  /bx x2 x1 sub 3 mul cx sub def
  /by y2 y1 sub 3 mul cy sub def
  /ax x3 x0 sub cx sub bx sub def
  /ay y3 y0 sub cy sub by sub def
  ax t 3 exp mul bx t t mul mul add cx t mul add x0 add
  ay t 3 exp mul by t t mul mul add cy t mul add y0 add
  3 ay t t mul mul mul 2 by t mul mul add cy add
  3 ax t t mul mul mul 2 bx t mul mul add cx add
  atan>

\pst@def{GetArms}<%
  /x1a armA AngleA cos mul x1 add def
  /y1a armA AngleA sin mul y1 add def
  /x2a armB AngleB cos mul x2 add def
  /y2a armB AngleB sin mul y2 add def>

\pst@def{NCCurve}<%
  \tx@GetPos
  x1 x2 sub y1 y2 sub \tx@Pyth
  2 div dup
  3 -1 roll mul /armA ED mul /armB ED
  \tx@GetArms
  x1a y1a x1 y1 tx@Dict begin ArrowA end
  x2a y2a x2 y2 tx@Dict begin ArrowB end
  curveto
  /LPutVar [ x1 y1 x1a y1a x2a y2a x2 y2 ] cvx def
  /LPutPos { t LPutVar \tx@BezierMidpoint } def>

\def\nccurve{\def\pst@par{}\pst@object{nccurve}}
\def\nccurve@i{\check@arrow{\nccurve@ii}}
\def\nccurve@ii#1#2{\nc@object{#1}{#2}{.5}{%
  /AngleA \psk@angleA\space def /AngleB \psk@angleB\space def
  \psk@ncurvB\space \psk@ncurvA\space
  \tx@NCCurve}}

\def\pccurve{\def\pst@par{}\pst@object{pccurve}}
\def\pccurve@i{\pc@object\nccurve@ii}
```

```
\def\ncarc{\def\pst@par{}\pst@object{ncarc}}
\def\ncarc@i{\check@arrow{\ncarc@ii}}
\def\ncarc@ii#1#2{\nc@object{#1}{#2}{.5}{%
  \tx@GetAngle dup
  \psk@arcangleA\space add /AngleA ED
  \psk@arcangleB\space sub 180 add /AngleB ED
  \psk@ncurvB\space \psk@ncurvA\space
  \tx@NCCurve}}

\def\pcarc{\def\pst@par{}\pst@object{pcarc}}
\def\pcarc@i{\pc@object\ncarc@ii}

\pst@def{AnglesMP}<%
  LPutVar
  t 3 gt
  { /t t 3 sub def }
  { t 2 gt
    { /t t 2 sub def 10 -2 roll }
    { t 1 gt
      { /t t 1 sub def 10 -4 roll }
      { 10 4 roll }
      ifelse }
    ifelse }
  ifelse
  6 { pop } repeat
  3 -1 roll exch \tx@LineMP>

\pst@def{NCAngles}<%
  \tx@GetPos
  \tx@GetArms
  /mtrx AngleA matrix rotate def
  x1a y1a mtrx transform pop
  x2a y2a mtrx transform exch pop
  mtrx itransform
  /y0 ED /x0 ED
  mark
  armB 0 ne { x2 y2 } if x2a y2a x0 y0 x1a y1a armA 0 ne { x1 y1 } if
  tx@Dict begin false \tx@Line end
  /LPutVar [ x2 y2 x2a y2a x0 y0 x1a y1a x1 y1 ] cvx def
  /LPutPos { \tx@AnglesMP } def>

\def\ncangles{\def\pst@par{}\pst@object{ncangles}}
\def\ncangles@i{\check@arrow{\ncangles@ii}}
\def\ncangles@ii#1#2{%
  \nc@object{#1}{#2}{1.5}{\ncangles@iii \tx@NCAngles}}
\def\ncangles@iii{%
  tx@Dict begin
  \ifdim\pslinearc>\z@
    /r \pst@number\pslinearc def
    /Lineto { \tx@Arcto } def
  \else
    /Lineto { L } def
  \fi
  end
  /AngleA \psk@angleA\space def /AngleB \psk@angleB\space def
  /armA \psk@armA\space def /armB \psk@armB\space def }
```

```
\def\pcangles{\def\pst@par{}\pst@object{pcangles}}
\def\pcangles@i{\pc@object\ncangles@ii}

\pst@def{NCAngle}<%
  \tx@GetPos
  /x2a armB AngleB cos mul x2 add def
  /y2a armB AngleB sin mul y2 add def
  /mtrx AngleA matrix rotate def
  x2a y2a mtrx transform pop
  x1 y1 mtrx transform exch pop
  mtrx itransform
  /y0 ED /x0 ED
  mark
  armB 0 ne { x2 y2 } if x2a y2a x0 y0 x1 y1
  tx@Dict begin false \tx@Line end
  /LPutVar [ x2 y2 x2 y2 x2a y2a x0 y0 x1 y1 ] cvx def
  /LPutPos { \tx@AnglesMP } def>

\def\ncangle{\def\pst@par{}\pst@object{ncangle}}
\def\ncangle@i{\check@arrow{\ncangle@ii}}
\def\ncangle@ii#1#2{%
  \nc@object{#1}{#2}{1.5}{\ncangles@iii \tx@NCAngle}}

\def\pcangle{\def\pst@par{}\pst@object{pcangle}}
\def\pcangle@i{\pc@object\ncangle@ii}

\pst@def{NCBar}<%
  \tx@GetPos
  \tx@GetArms
  /mtrx AngleA matrix rotate def
  x1a y1a mtrx transform pop
  x2a y2a mtrx transform pop sub
  dup 0 mtrx itransform
  3 -1 roll 0 gt
  { /y2a exch y2a add def /x2a exch x2a add def }
  { /y1a exch neg y1a add def /x2a exch neg x2a add def }
  ifelse
  mark
  x2 y2 x2a y2a x1a y1a x1 y1
  tx@Dict begin false \tx@Line end
  /LPutVar [ x2 y2 x2 y2 x2a y2a x1a y1a x1 y1 ] cvx def
  /LPutPos { LPutVar \tx@AnglesMP } def>

\def\ncbar{\def\pst@par{}\pst@object{ncbar}}
\def\ncbar@i{\check@arrow{\ncbar@ii}}
\def\ncbar@ii#1#2{\nc@object{#1}{#2}{1.5}{%
  \ncangles@iii /AngleB \psk@angleA def \tx@NCBar}}

\def\pcbar{\def\pst@par{}\pst@object{pcbar}}
\def\pcbar@i{\pc@object\ncbar@ii}

\pst@def{NCDiag}<%
  \tx@GetPos
  \tx@GetArms
  mark
  x2 y2 x2a y2a x1a y1a x1 y1
```

```
  tx@Dict begin false \tx@Line end
  /LPutVar [ x2 y2 x2 y2 x2a y2a x1a y1a x1 y1 ] cvx def
  /LPutPos { \tx@AnglesMP } def>

\def\ncdiag{\def\pst@par{}\pst@object{ncdiag}}
\def\ncdiag@i{\check@arrow{\ncdiag@ii}}
\def\ncdiag@ii#1#2{%
  \nc@object{#1}{#2}{1.5}{\ncangles@iii \tx@NCDiag}}
```

\pcdiag

```
\def\pcdiag{\def\pst@par{}\pst@object{pcdiag}}
\def\pcdiag@i{\pc@object\ncdiag@ii}

\pst@def{NCDiagg}<%
  OffsetA AngleA NodesepA nodeA \tx@GetEdge
  /y1 ED /x1 ED
  /x1a armA AngleA cos mul x1 add def
  /y1a armA AngleA sin mul y1 add def
  nodeB \tx@GetCenter
  y1a sub exch x1a sub \tx@Atan 180 add /AngleB ED
  OffsetB AngleB NodesepB nodeB \tx@GetEdge
  /y2 ED /x2 ED
  mark
  x2 y2 x1a y1a x1 y1
  tx@Dict begin false \tx@Line end
  /LPutVar [ x2 y2 x2 y2 x2 y2 x1a y1a x1 y1] cvx def
  /LPutPos { \tx@AnglesMP } def>

\def\ncdiagg{\def\pst@par{}\pst@object{ncdiagg}}
\def\ncdiagg@i{\check@arrow{\ncdiagg@ii}}
\def\ncdiagg@ii#1#2{%
  \nc@object{#1}{#2}{.5}{\ncangles@iii \tx@NCDiagg}}
```

\pcdiagg

```
\def\pcdiagg{\def\pst@par{}\pst@object{pcdiagg}}
\def\pcdiagg@i{\pc@object\ncdiagg@ii}
```

\tx@LoopMP

```
\pst@def{LoopMP}<%
  /t t abs def
  [ LPutVar ] length 2 div 1 sub dup t lt { /t ED } { pop } ifelse
  mark LPutVar
  t cvi { /t t 1 sub def pop pop } repeat
  counttomark 1 add 4 roll cleartomark
  3 -1 roll exch \tx@LineMP>

\pst@def{NCLoop}<%
  \tx@GetPos
  \tx@GetArms
  /mtrx AngleA matrix rotate def
  x1a y1a mtrx transform loopsize add /y1b ED /x1b ED
  /x2b x2a y2a mtrx transform pop def
  x2b y1b mtrx itransform /y2b ED /x2b ED
  x1b y1b mtrx itransform /y1b ED /x1b ED
```

```
  mark
  armB 0 ne { x2 y2 } if x2a y2a x2b y2b x1b y1b x1a y1a armA
    0 ne { x1 y1 } if
  tx@Dict begin false \tx@Line end
  /LPutVar [ x2 y2 x2a y2a x2b y2b x1b y1b x1a y1a x1 y1 ] cvx def
  /LPutPos { \tx@LoopMP } def>
```

\psset@loopsize

```
\def\psset@loopsize#1{\pst@getlength{#1}\psk@loopsize}
\psset@loopsize{1cm}

\def\ncloop{\def\pst@par{}\pst@object{ncloop}}
\def\ncloop@i{\check@arrow{\ncloop@ii}}
\def\ncloop@ii#1#2{%
  \nc@object{#1}{#2}{2.5}%
    {\ncangles@iii /loopsize \psk@loopsize\space def \tx@NCLoop}}

\def\pcloop{\def\pst@par{}\pst@object{pcloop}}
\def\pcloop@i{\pc@object\ncloop@ii}
```

\tx@NCCircle

```
\pst@def{NCCircle}<%
  nodeA \tx@GetCenter
  0 0 NodesepA nodeA \tx@GetEdge
  % Stack:  x-center y-center x-edge y-origin
  pop 3 1 roll
  /Y ED /X ED    % center
  X sub 2 div    % half distance to edge
  dup 2 exp r r mul sub abs sqrt atan 2 mul /a ED % angle to edge
  r AngleA 90 add \tx@PtoC      % displacement to origin
  Y add exch X add exch         % origin
  2 copy /LPutVar [ 4 2 roll r a ] def
  /LPutPos { LPutVar aload pop t 360 mul add dup 5 1 roll
    90 sub \tx@PtoC 3 -1 roll add 3 1 roll add exch 3 -1 roll } def
  r
  AngleA 90 sub a add           % begin arc angle
  AngleA 270 add a sub          % end arc angle
  % Stack:  x0 y0 r a1 a2
  tx@Dict begin
    /angleB ED
    /angleA ED
    /r ED
    /c 57.2957 r \tx@Div def
    /y ED
    /x ED>

\def\nccircle{\def\pst@par{}\pst@object{nccircle}}
\def\nccircle@i{\check@arrow{\nccircle@ii}}
\def\nccircle@ii#1#2{%
  \pssetlength\pst@dima{#2}%
  \nc@object{#1}{#1}{.5}{%
    /AngleA \psk@angleA def
    /r \pst@number\pst@dima def
     \tx@NCCircle \psarc@v end}}
```

44 Node Labels

\pst@getlputrot

```
\def\pst@getlputrot#1{%
  \@ifnextchar(%
    {\def\pst@rot{}#1}%
    {\pst@@getlputrot{\@ifnextchar({#1}{#1(\lputpos@default)}}}}
\def\pst@@getlputrot#1#2{%
  \pst@expandafter{\@ifnextchar:{\pst@@@getlputrot}%
    {\@ifstar{\pst@@@getrot}{\pst@@getrot}}}{#2}\@nil
  \ifx\pst@rotlist\@empty\else
    \edef\pst@rotlist{\pst@rotlist \pst@rot add }%
  \fi
  #1}
\def\pst@@@getlputrot#1#2\@nil{%
  \pst@@getrot#2\@nil
  \edef\pst@rot{langle \ifx\pst@rot\@empty\else\pst@rot add \fi}}%
```

LPutCoor

```
\pst@def{LPutCoor}<%
  tx@NodeDict /LPutPos known
  { gsave
      LPutPos
      tx@Dict begin
        /langle ED
        CM 3 1 roll
        \tx@STV
        CP 3 -1 roll sub neg 3 1 roll sub exch
        moveto
        setmatrix
        CP
      end
    grestore }
  { 0 0 tx@Dict /langle 0 def end }
  ifelse>
```

\psput@lput

```
\def\psput@lput#1#2{%
  \pst@checknum{#1}\pst@tempa
  \hbox{%
    \pst@Verb{%
      \pst@nodedict
        /t \pst@tempa\space def
        \tx@LPutCoor
      end
      \tx@PutBegin}%
    \box#2%
    \pst@Verb{\tx@PutEnd}}}
```

\lput

```
\def\lput{\begin@psput{\pst@getref{\pst@getlputrot{\end@psput\lput@i}}}}
\def\lput@i(#1){%
```

```
    \pst@makesmall\pst@hbox
    \ifx\pst@rot\@empty\else\pst@rotate\pst@hbox\fi
    \psput@lput{#1}\pst@hbox}
```

\mput

```
\def\mput{%
  \begin@psput{\def\pst@rot{}\pst@getref{\end@psput\lput@i(\lputpos@default)}}}
```

\aput, \Aput, \bput, \Bput

```
\def\aput@#1{\begin@psput{%
  \def\pst@refangle{#1 }%
  \@ifnextchar[{\aput@i}{\pst@getlputrot{\end@psput\aput@ii}}}}
\def\aput@i[#1]{%
  \pssetlength\pslabelsep{#1}\pst@getlputrot{\end@psput\aput@ii}}
\def\aput@ii(#1){%
    \uput@iv\aput@iii
    \psput@lput{#1}\pst@hbox}
\def\aput@iii{exch pop add a \tx@PtoC h1 add exch w1 add exch }
\def\aput{\aput@{langle 90 add}}
\def\bput{\aput@{langle 90 sub}}
\def\Aput@#1{\begin@psput{%
  \def\pst@refangle{#1 }%
  \def\pst@rot{}%
  \@ifnextchar[{\Aput@i}{\end@psput\aput@ii(\lputpos@default)}}}
\def\Aput@i[#1]{%
  \pssetlength\pslabelsep{#1}%
  \end@psput\aput@ii(\lputpos@default)}
\def\Aput{\Aput@{langle 90 add}}
\def\Bput{\Aput@{langle 90 sub}}
```

\Lput, \Mput

These are obsolete.

```
\def\Lput{%
  \begin@psput{\pst@getlabelsep{\pst@getlputrot{\end@psput{\Rput@i\lput@i}}}}}
\def\Mput{%
  \begin@psput{%
    \def\pst@rot{}%
    \pst@getlabelsep{\end@psput{\Rput@i\lput@i}(\lputpos@default)}}}
```

45 Node coordinates

\node@coor

```
\def\node@coor#1;#2\@nil{%
  \pst@getnode{#1}\pst@tempg
  \edef\pst@coor{%
    \pst@nodedict
    tx@NodeDict \pst@tempg known
    { \pst@tempg load \tx@GetCenter }
    { 0 0 }
    ifelse
```

```
    end }}
```

\Node@coor

```
\def\Node@coor[#1]#2;#3\@nil{%
  \begingroup
    \psset{#1}%
    \pst@getnode{#2}\pst@tempg
    \xdef\pst@tempg{%
      \pst@nodedict
      tx@NodeDict \pst@tempg known
      { \psk@offsetA \psk@angleA \psk@nodesepA \pst@tempg load \tx@GetEdge }
      { 0 0 }
      ifelse
      end }%
  \endgroup
  \let\pst@coor\pst@tempg}
\pst@ATH<end>
\catcode`\@=\TheAtCode\relax
\endinput
```

Index

The **bold** numbers denote the pages where the entries are defined, and all other numbers indicate the *lines of code* where the entries are used.

Q

R

S

T

U

W

Y