

# The `pict2e` package\*

Hubert Gäßlein, Rolf Niepraschk<sup>†</sup> and Josef Tkadlec<sup>‡</sup>

2020/08/16

## Abstract

This package was described in the 2nd edition of “L<sup>A</sup>T<sub>E</sub>X: A Document Preparation System”, but the L<sup>A</sup>T<sub>E</sub>X project team declined to produce the package. For a long time, L<sup>A</sup>T<sub>E</sub>X has included a “`pict2e` package” that merely produced an apologetic error message.

The new package extends the existing L<sup>A</sup>T<sub>E</sub>X `picture` environment, using the familiar technique (cf. the `graphics` and `color` packages) of driver files. In the user-level part of this documentation there is a fair number of examples of use, showing where things are improved by comparison with the Standard L<sup>A</sup>T<sub>E</sub>X `picture` environment.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Usage</b>	<b>3</b>
2.1	Package options	3
2.1.1	Driver options	3
2.1.2	Other options	3
2.1.3	Debugging options	3
2.2	Configuration file	4
2.3	Details: Changes to user-level commands	4
2.3.1	Line	4
2.3.2	Vector	5
2.3.3	Circle and Dot	6
2.3.4	Oval	7
2.3.5	Bezier Curves	8
2.4	Extensions	9
2.4.1	Circle arcs	9
2.4.2	Line, Vector, polyline, polyvector, and polygon	9
2.4.3	Path commands	9
2.4.4	Ends of paths, joins of subpaths	10
<b>3</b>	<b>Implementation</b>	<b>13</b>
3.1	Initialisation	13
3.2	Preliminaries	13
3.3	Option processing	14
3.4	Output driver check	15
3.5	Mode check	16
3.6	Graphics operators	17
3.7	Low-level operations	17

---

\*This document corresponds to `pict2e.sty` v0.4a, dated 2020/08/16, documentation dated 2020/08/16.

<sup>†</sup>Rolf.Niepraschk@gmx.de

<sup>‡</sup>j.tkadlec@email.cz

3.7.1	Collecting the graphics instructions and handling the output	17
3.7.2	Auxilliary macros	18
3.8	Medium-level operations	18
3.8.1	Transformations	18
3.8.2	Path definitions	19
3.9	“Pythagorean Addition” and Division	20
3.10	High-level operations	23
3.10.1	Line	23
3.10.2	Vector	24
3.10.3	Circle and Dot	27
3.10.4	Oval	28
3.10.5	Quadratic Bezier Curve	30
3.10.6	Circle arcs	31
3.10.7	Line, Vector, polyline, polyvector, and polygon	33
3.10.8	Path commands	34
3.10.9	Ends of paths, joins of subpaths	34
3.11	Commands from other packages	35
3.11.1	Package <code>ebezier</code>	35
3.11.2	Other packages	35
3.12	Mode ‘original’	35
3.13	Final clean-up	36

## List of Figures

1	Line	5
2	Vector	6
3	Vector: shape variants of the arrow-heads	7
4	Circle and Dot	7
5	Oval: Radius argument for <code>\oval</code> vs. <code>\maxovalrad</code>	8
6	Oval: Radius argument for <code>\oval</code> : length vs. number	11
7	Quadratic Bezier curves	12
8	Cubic Bezier curves	12
9	Quadratic (green) and Cubic Bezier curves	12
10	$\text{\LaTeX}$ -like implementation of <code>\vector</code>	25
11	PSTricks-like implementation of <code>\vector</code>	26
12	Auxillary macro <code>\pIIE@qcircle</code> —draw a quarter circle	28

## 1 Introduction

Here’s a quote from the obsolete original official version of the `pict2e` package (1993–2003):

*The package `pict2e` that is mentioned in the 2nd edition of “ $\text{\LaTeX}$ : A Document Preparation System” has not yet been produced. It is unlikely that the  $\text{\LaTeX}$ 3 Project Team will ever produce this package thus we would be very happy if someone else creates it.*

: -) Finally, someone has produced a working implementation of the `pict2e` package.

This package redefines some of the drawing commands of the  $\text{\LaTeX}$  picture environment. Like the `graphics` and `color` packages, it uses driver files.

Currently there are only back-ends for PostScript and PDF. (Other output formats may be added in the future.)

Note/Warning:

- Documentation has been written somewhat “hastily” and may be inaccurate.
- The status of this package is currently somewhere between “beta” and “release” ... Users and package programmers should *not* rely on *any* feature sported by the internal commands. (Especially, the internal control sequence names may change without notice in future versions of this package.)

## 2 Usage

To use the `pict2e` package, you put a `\usepackage[optionlist]{pict2e}` instruction in the preamble of your document. Likewise, class or package writers just say `\RequirePackage[optionlist]{pict2e}` in an appropriate place in their class or package file. (Nothing unusual here.)

Like the `graphics` and `color` packages, the `pict2e` package supports a configuration file (see Section 2.2).

### 2.1 Package options

#### 2.1.1 Driver options

driver	notes	driver	notes
dvips	x	oztex	(x)
xdvi	x	dvipsone	x?
pdftex	x	dviwindo	x?
vtex	x	dvipdf	x?
dvipdfm	x	textures	x?
dvipdfmx	x	pctexps	x?
xetex	x	pctex32	x?
luatex (> 0.85)	x		

x = supported; (x) = supported but untested;

x? = not yet implemented

The driver options are (mostly) implemented by means of definition files (`p2e-driver.def`). For details, see file `p2e-drivers.dtx`.

*Note: You should specify the same driver for `pict2e` you use with the `graphics/x` and `color` packages. Otherwise, things may go haywire.*

#### 2.1.2 Other options

Currently, there are two options that allow you to choose between variants of the arrows-heads generated by the `\vector` command. See Figure 3 in Section 2.3.2 for the difference.

option	meaning
<code>ltxarrows</code>	Draw L <sup>A</sup> T <sub>E</sub> X style vectors (default).
<code>pstarrows</code>	Draw P <sup>S</sup> Tricks style vectors.

#### 2.1.3 Debugging options

These options are (mainly) for development and testing purposes.

option	meaning
<code>original</code>	Suppresses the new definitions.
<code>debug</code>	Suppresses the compressing of pdfT <sub>E</sub> X output; marks the <code>pict2e</code> generated code in the output files.
<code>hide</code>	Suppresses all graphics output from <code>pict2e</code> .

## 2.2 Configuration file

Similar to the `graphics` and `color` packages, in most cases it is not necessary to give a driver option explicitly with the `\usepackage` (or `\RequirePackage`) command, if a suitable configuration file `pict2e.cfg` is present on your system (see the example file `pict2e-example.cfg`). On many systems it may be sufficient to copy `pict2e-example.cfg` to `pict2e.cfg`; on others you might need to modify your copy to suit your system.

## 2.3 Details: Changes to user-level commands

This section describes the improvements of the new implementation of (some of) the `picture` commands. For details, look up “`pict2e` package” in the index of the  $\text{\LaTeX}$  manual [1].

Here’s a collection of quotes relevant to the `pict2e` package from the  $\text{\LaTeX}$  manual [1].

From [1, p. 118]:

*However, the `pict2e` package uses device-driver support to provide enhanced versions of these commands that remove some of their restrictions. The enhanced commands can draw straight lines and arrows of any slope, circles of any size, and lines (straight and curved) of any thickness.*

From [1, p. 179]:

*`pict2e` Defines enhanced versions of the `picture` environment commands that remove restrictions on the line slope, circle radius, and line thickness.*

From [1, pp. 221–223]:

**`\qbezier`**

*(With the `pict2e` package, there is no limit to the number of points plotted.)*

**`\line` and `\vector`** Slopes  $|x|, |y| \leq 6$  or 4, with no common divisor except  $\pm 1$ :

*(These restrictions are eliminated by the `pict2e` package.)*

**`\line` and `\vector`** Smallest horizontal extent of sloped lines and vectors that can be drawn:

*(This does not apply when the `pict2e` package is loaded.)*

**`\circle` and `\circle*`** Largest circles and disks that can be drawn:

*(With the `pict2e` package, any size circle or disk can be drawn.)*

**`\oval`** [ $\langle \text{rad} \rangle$ ]:

*An explicit rad argument can be used only with the `pict2e` package; the default value is the radius of the largest quarter-circle  $\text{\LaTeX}$  can draw without the `pict2e` package.*

### 2.3.1 Line

**`\line`** `\line( $\langle X, Y \rangle$ ){ $\langle LEN \rangle$ }`

In the Standard  $\text{\LaTeX}$  implementation the slope arguments ( $\langle X, Y \rangle$ ) are restricted to integers in the range  $-6 \leq X, Y \leq +6$ , with no common divisors except  $\pm 1$ . (I.e.,  $X$  and  $Y$  must be relatively prime.) Furthermore, only horizontal and vertical lines can assume arbitrary thickness; sloped lines are restricted to the widths given by the `\thinlines` and `\thicklines` declarations (i.e., 0.4pt and 0.8pt, respectively).

From [1, p. 222]:

*These restrictions are eliminated by the `pict2e` package.*

However, to avoid overflow of  $\text{\TeX}$ ’s `dimens`, the slope arguments are real numbers in the range  $-16383 \leq X, Y \leq +16383$ . It is usually not a good idea to use slope arguments with the absolute value less than  $10^{-4}$  (the best accuracy is obtained if you use multiples of arguments

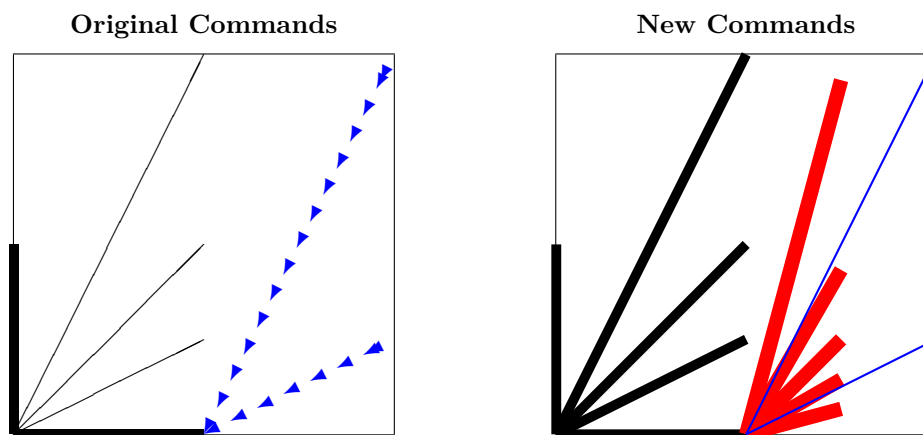


Figure 1: Line

such that you eliminate as much decimal parts as possible). The slope greater then 16384 cannot be obtained.

Furthermore, unlike the Standard  $\text{\LaTeX}$  implementation, which silently converts the “impossible” slope to a vertical line extending in the upward direction  $((0,0) \mapsto (0,1))$ , the `pict2e` package now treats this as an error.

In the Standard  $\text{\LaTeX}$  implementation the horizontal extent of sloped lines must be at least 10 pt.

From [1, p. 222]:

*This does not apply when the `pict2e` package is loaded.*

Figure 1 shows the difference between the old and new implementations: The black lines in the left half of each picture all have slopes that conform to the restrictions of Standard  $\text{\LaTeX}$ . However, with the new implementation of `pict2e` sloped lines may assume any arbitrary width given by the `\linethickness` declaration. The right half demonstrates that now arbitrary slopes are possible.

The blue lines represent “illegal” slopes specifications, i.e., with common divisors. Note the funny effect Standard  $\text{\LaTeX}$  produces in such cases. (In  $\text{\LaTeX}$  releases prior to 2003/12/01, some such “illegal” slopes might even lead to infinite loops! Cf. problem report latex/3570.)

The new implementation imposes no restriction with respect to line thickness, minimal horizontal extent, and slope.

The red lines correspond to angles of  $15^\circ$ ,  $30^\circ$ ,  $45^\circ$ ,  $60^\circ$ , and  $75^\circ$ , respectively. This was achieved by multiplying the sine and cosine of each angle by 1000 and rounding to the nearest integer, like this:

```
\put(50,0){\line(966,259){25}}
\put(50,0){\line(866,500){25}}
\put(50,0){\line(707,707){25}}
\put(50,0){\line(500,866){25}}
\put(50,0){\line(259,966){25}}
```

### 2.3.2 Vector

`\vector` `\vector(<X,Y>){<LEN>}`

In the Standard  $\text{\LaTeX}$  implementation the slope arguments  $(\langle X,Y \rangle)$  are restricted to integers in the range  $-4 \leq X,Y \leq +4$ , with no common divisors except  $\pm 1$ . (I.e.,  $X$  and  $Y$  must be relatively prime.) Furthermore, arrow heads come only in two shapes, corresponding to the `\thinlines` and `\thicklines` declarations. (There’s also a flaw: the lines will be printed over the arrow heads. See vertical vector in Figure 2.)

From [1, p. 222]:

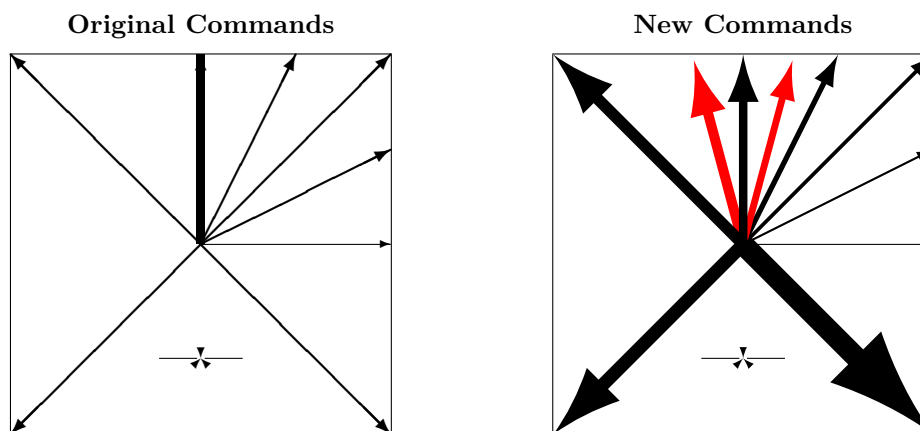


Figure 2: Vector

*These restrictions are eliminated by the `pict2e` package.*

However, to avoid overflow of  $\text{T}_{\text{E}}\text{X}$ 's dimen arithmetic, the current implementation restricts the slope arguments to real numbers in the range  $-1000 \leq X, Y \leq +1000$ , which should be enough. It is usually not a good idea to use slope arguments with the absolute value less than  $10^{-4}$  (the best accuracy is obtained if you use multiples of arguments such that you eliminate as much decimal parts as possible). The slope greater than 16384 cannot be obtained.

Furthermore, unlike the Standard  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  implementation, which silently converts the “impossible” slope to a vertical vector extending in the upward direction  $((0, 0) \mapsto (0, 1))$ , the `pict2e` package now treats this as an error.

In the Standard  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  implementation the horizontal extent of sloped vectors must be at least 10 pt.

From [1, p. 222]:

*This does not apply when the `pict2e` package is loaded.*

Figure 2 shows the difference between the old and new implementations: The black arrows all have “legal” slopes. The red arrows have slope arguments out of the range permitted by Standard  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ . Slope arguments that are “illegal” in Standard  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  produce results similar to those with the `\line` command (this has not been demonstrated here).

The new implementation imposes no restriction with respect to line thickness, minimal horizontal extent, and slope.

As with Standard  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ , the arrow head will always be drawn. In particular, only the arrow head will be drawn, if the total length of the arrow is less than the length of the arrow head. See right hand side of Figure 3.

The current version of the `pict2e` package offers two variants for the shape of the arrow heads, controlled by package options. One variant tries to mimic the fonts used in the Standard  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  implementation (package option `ltxarrows`, the default; see Figure 3, top row), though it is difficult to extrapolate from just two design sizes. The other one is implemented like the arrows of the `PSTricks` package [8] (package option `pstarrows`; see Figure 3, bottom row).

### 2.3.3 Circle and Dot

```
\circle \circle{<DIAM>}
\circle* \circle*{<DIAM>}
```

The (hollow) circles and disks (filled circles) of the Standard  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  implementation had severe restrictions on the number of different diameters and maximum diameters available.

From [1, p. 222]:

*With the `pict2e` package, any size circle or disk can be drawn.*

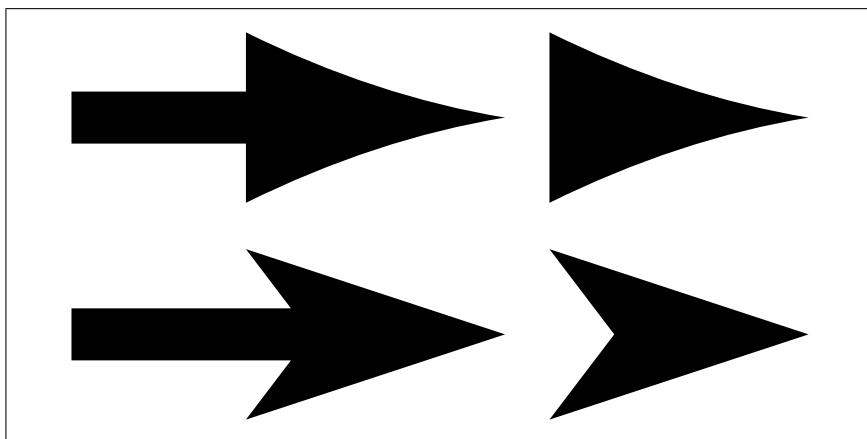


Figure 3: Vector: shape variants of the arrow-heads. Top: L<sup>A</sup>T<sub>E</sub>X style vectors. Bottom: PSTricks style vectors.

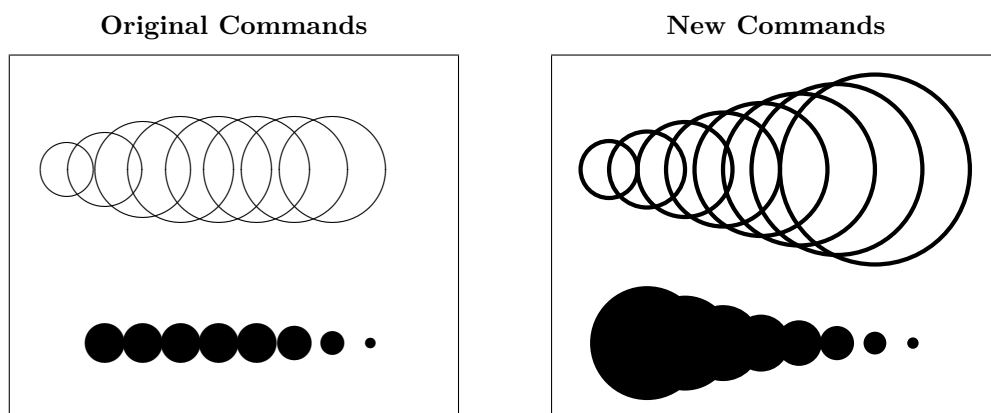


Figure 4: Circle and Dot

With the new implementation there are no more restrictions to the diameter argument. (However, negative diameters are now trapped as an error.)

Furthermore, hollow circles (like sloped lines) can now be drawn with any line thickness. Figure 4 shows the difference.

### 2.3.4 Oval

`\oval` `\oval[ $\langle rad \rangle$ ]( $\langle X, Y \rangle$ )[ $\langle POS \rangle$ ]`

In the Standard L<sup>A</sup>T<sub>E</sub>X implementation, the user has no control over the shape of an oval besides its size, since its corners would always consist of the “quarter circles of the largest possible radius less than or equal to *rad*” [1, p. 223].

From [1, p. 223]:

*An explicit rad argument can be used only with the `pict2e` package; the default value is the radius of the largest quarter-circle L<sup>A</sup>T<sub>E</sub>X can draw without the `pict2e` package.*

This default value is 20 pt, a length. However, in an early reimplementaion of the picture commands [5], there is such an optional argument too, but it is given as a mere number, to be multiplied by `\unitlength`.

Since both alternatives may make sense, we left the choice to the user. (See Figure 6 for the differences.) I.e., this implementation of `\oval` will “auto-detect” whether its [ $\langle rad \rangle$ ]

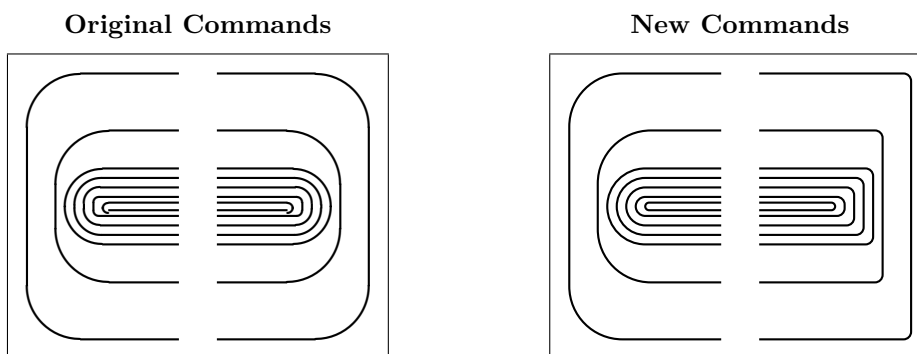


Figure 5: Oval: Radius argument for `\oval` vs. `\maxovalrad`

`\maxovalrad` argument is a length or a number. Furthermore, the default value is not hard-wired either; the user may access it under the moniker `\maxovalrad`, by the means of `\renewcommand*`. (Names or values of length and counter registers may be given as well, both as an explicit [ $\langle rad \rangle$ ] argument and when redefining `\maxovalrad`.)

(Both [ $\langle rad \rangle$ ] and the default value `\maxovalrad` are ignored in “standard L<sup>A</sup>T<sub>E</sub>X mode”).

The behaviour of `\oval` in the absence of the [ $\langle rad \rangle$ ] argument is shown in Figure 5, left half of each picture. Note that in the Standard L<sup>A</sup>T<sub>E</sub>X implementation there is a minimum radius as well (innermost “salami” is “broken”). In the right half of each picture, a [ $\langle rad \rangle$ ] argument has been used: it has no effect with the original `\oval` command.

Both [ $\langle rad \rangle$ ] and `\maxovalrad` may be given as an explicit (rigid) length (i.e., with unit) or as a number. In the latter case the value is used as a factor to multiply by `\unitlength`. (A length or counter register will do as well, of course.)

If a number is given, the rounded corners of an oval will scale according to the current value of `\unitlength`. (See Figure 6, first row.)

If a length is specified, the rounded corners of an oval will be the same regardless of the current value of `\unitlength`. (See Figure 6, second row.)

The default value is 20 pt as specified for the [ $\langle rad \rangle$ ] argument of `\oval` by the L<sup>A</sup>T<sub>E</sub>X manual [1, p. 223]. (See Figure 6, third row.)

### 2.3.5 Bezier Curves

`\bezier` `\bezier{ $\langle N \rangle$ }( $\langle AX,AY \rangle$ )( $\langle BX,BY \rangle$ )( $\langle CX,CY \rangle$ )`

`\qbezier` `\qbezier[ $\langle N \rangle$ ]( $\langle AX,AY \rangle$ )( $\langle BX,BY \rangle$ )( $\langle CX,CY \rangle$ )`

`\cbezier` `\cbezier[ $\langle N \rangle$ ]( $\langle AX,AY \rangle$ )( $\langle BX,BY \rangle$ )( $\langle CX,CY \rangle$ )( $\langle DX,DY \rangle$ )`

`\qbeziermax` In Standard L<sup>A</sup>T<sub>E</sub>X, the  $N$  argument specifies the number of points to plot:  $N + 1$  for a positive integer  $N$ , appropriate number (at most `\qbeziermax`) for  $N = 0$  or if the optional argument is missing. With L<sup>A</sup>T<sub>E</sub>X versions prior to 2003/12/01, the quadratic Bezier curves plotted by this package will not match those of the Standard L<sup>A</sup>T<sub>E</sub>X implementation exactly, due to a bug in positioning the dots used to produce a curve (cf. latex/3566).

`\bezier` is the obsolescent variant from the old `bezier` package of vintage L<sup>A</sup>T<sub>E</sub>X2.09.

The `\cbezier` command draws a cubic Bezier curve; see [3]. (This is not mentioned in [1] and has been added to the package deliberately.)

From [1, p. 221–223]:

*With the `pict2e` package, there is no limit to the number of points plotted.*

More accurately, if the optional argument is absent or is 0, the `pict2e` package uses primitive operators of the output (back-end) format to draw a full curve.



## 2.4 Extensions

This section describe new commands that extend the possibilities of the `picture` environment. It is not our aim to create a powerful collection of macros (like `pstricks` or `tikz`). The main goal of this package is to eliminate the limitations of the standard `picture` commands. But this is done by PostScript and PDF operators that might be easily used for user-level commands and hence significantly improve the drawing possibilities.

### 2.4.1 Circle arcs

```
\arc \arc[⟨ANGLE1,ANGLE2⟩]{⟨RAD⟩}
\arc* \arc*[⟨ANGLE1,ANGLE2⟩]{⟨RAD⟩}
```

These commands are generalizations of `\circle` and `\circle*` commands except that the radius instead of the diameter is given. The optional argument is a comma separated pair of angles given in degrees (implicit value is  $[0, 360]$ ). The arc starts at the point given by *ANGLE1*. If *ANGLE2* is greater than *ANGLE1* the arc is drawn in the positive orientation (anticlockwise), if the *ANGLE2* is smaller than *ANGLE1* the arc is drawn in the negative orientation (clockwise). The angle of the arc is the absolute value the difference of *ANGLE1* and *ANGLE2*. Hence the pair  $[-10, 80]$  gives the same arc as  $[80, -10]$  (a quarter of a circle) while the pairs  $[80, 350]$  and  $[350, 80]$  give the complementary arc.

In fact, the arc is approximated by cubic Bezier curves with an inaccuracy smaller than 0.0003 (it seems to be sufficiently good).

If `\squarecap` is active then `\arc{⟨RAD⟩}` produces a circle with a square.

An equivalent `\pIIearc` to `\arc` is defined to solve possible conflicts with other packages.

### 2.4.2 Line, Vector, polyline, polyvector, and polygon

```
\Line \Line(⟨X1,Y1⟩)(⟨X2,Y2⟩)
\polyline \polyline(⟨X1,Y1⟩)(⟨X2,Y2⟩)...(⟨Xn,Yn⟩)
\Vector \Vector(⟨X1,Y1⟩)(⟨X2,Y2⟩)
\polyvector \polyvector(⟨X1,Y1⟩)(⟨X2,Y2⟩)...(⟨Xn,Yn⟩)
\polygon \polygon(⟨X1,Y1⟩)(⟨X2,Y2⟩)...(⟨Xn,Yn⟩)
\polygon* \polygon*(⟨X1,Y1⟩)(⟨X2,Y2⟩)...(⟨Xn,Yn⟩)
```

A natural way how to describe a line segment is to give the coordinates of the endpoints. The syntax of the `\line`/`\vector` is different because the lines in the standard `picture` environment are made from small line segments of a limited number of slopes given in a font. However, this package changes the `\line` command computing the coordinates of the endpoints and using an internal macro for drawing a line segment with given endpoints. Hence it would be crazy do not use this possibility directly. This is done by the commands `\Line` and `\Vector`. The commands `\polyline` and `\polyvector` draws a stroken line/vector connecting points with given coordinates. The command `\polygon` draws a polygon with given vertices, the star variant gives filled polygon. At least two points should be given.

These command need not be used within a `\put` command (if the coordinates are absolute).

### 2.4.3 Path commands

```
\moveto \moveto(⟨X,Y⟩)
\lineto \lineto(⟨X,Y⟩)
\curveto \curveto(⟨X2,Y2⟩)(⟨X3,Y3⟩)(⟨X4,Y4⟩)
\circlearc \circlearc[⟨N⟩]{⟨X⟩}{⟨Y⟩}{⟨RAD⟩}{⟨ANGLE1⟩}{⟨ANGLE2⟩}
```

These commands directly correspond to the PostScript and PDF path operators. You start defining a path giving its initial point by `\moveto`. Then you can consecutively add a line segment to a given point by `\lineto`, a cubic Bezier curve by `\curveto` (two control points and the endpoint are given) or an arc by `\circlearc` (mandatory parameters are coordinates of the center, radius, initial and final angle).

Drawing arcs is a bit more complicated. There is a special operator only in PostScript (not in PDF) but also in PostScript it is approximated by cubic Bezier curves. Here we use common definition for PostScript and PDF. The arc is drawn such that the initial point given by the initial angle is rotated by  $ANGLE2 - ANGLE1$  (anticlockwise for positive value and clockwise for negative value) after reducing this difference to the interval  $[-720, 720]$ . Implicitly (the optional parameter  $N = 0$ ) before drawing an arc a `\lineto` to the initial point of the arc is added. For  $N = 1$  `\moveto` instead of `\lineto` is executed—it is useful if you start the path by an arc and do not want to compute and set the initial point. For  $N = 2$  the `\lineto` before drawing the arc is omitted—it leads to a bit shorter code for the path but you should be sure that the already defined part of the path ends precisely at the initial point of the arc.

`\closepath` The command `\closepath` is equivalent to `\lineto` to the initial point of the path. After defining paths you might use either `\strokepath` to draw them or, for closed paths, `\fillpath` to draw an area bounded by them.

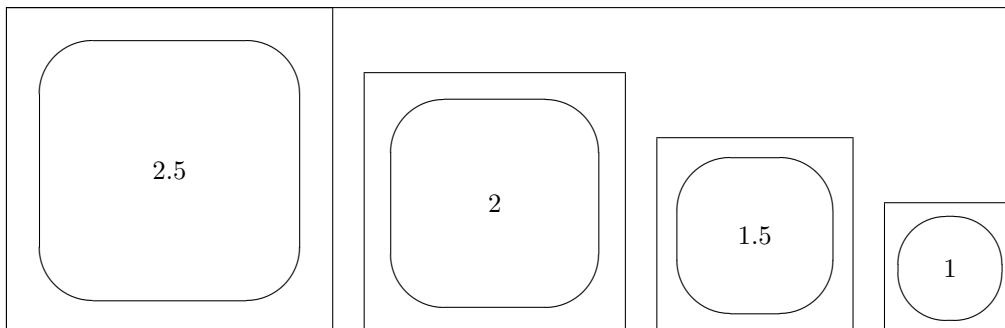
The path construction need not be used within a `\put` command (if the coordinates are absolute).

#### 2.4.4 Ends of paths, joins of subpaths

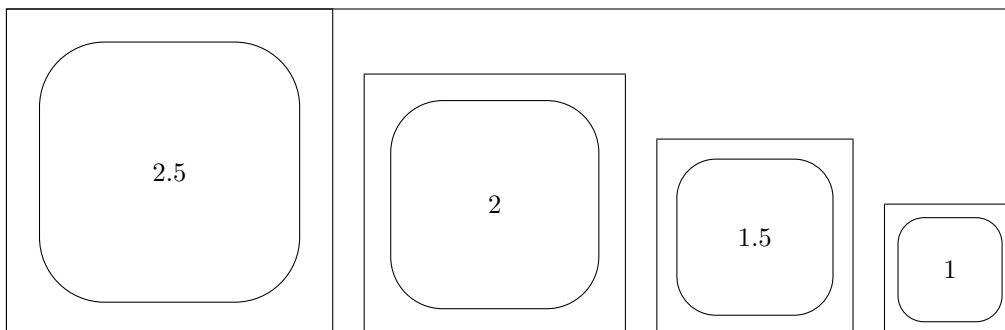
`\buttcap` The shape of ends of paths is controlled by the following commands: `\buttcap` (implicit) define the end as a line segment, `\roundcap` adds a halfdisc, `\squarecap` adds a halfsquare. `\squarecap` While `\squarecap` is ignored for the path with zero length, `\roundcap` places a disc to the given point. These commands do not apply to `\vector` and to closed paths (`\circle`, full `\oval`, parameter, path constructions ended by `\closepath`).

`\miterjoin` The shape of joins of subpaths is controlled by the following commands: `\miterjoin` (implicit) might be defined in such a way that “boundaries” of subpaths are prolonged until they intersect (it might be a rather long distance for lines with a small angle between them); `\roundjoin` corresponds to `\roundcap` for both subpaths; `\beveljoin` adds a convex hull of terminal line segments of both subpaths.

**Original Commands,  $[\langle rad \rangle]$  or `\maxovalrad` ignored**



**New Commands,  $[\langle rad \rangle]$  or `\maxovalrad` depends on `\unitlength`**



**New Commands,  $[\langle rad \rangle]$  or `\maxovalrad` a fixed length**

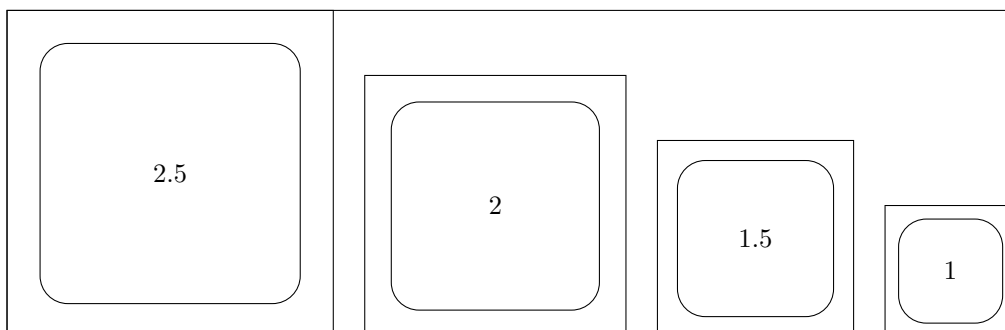


Figure 6: Oval: Radius argument for `\oval`: length vs. number. The number at the centre of each oval gives the relative value of `\unitlength`.

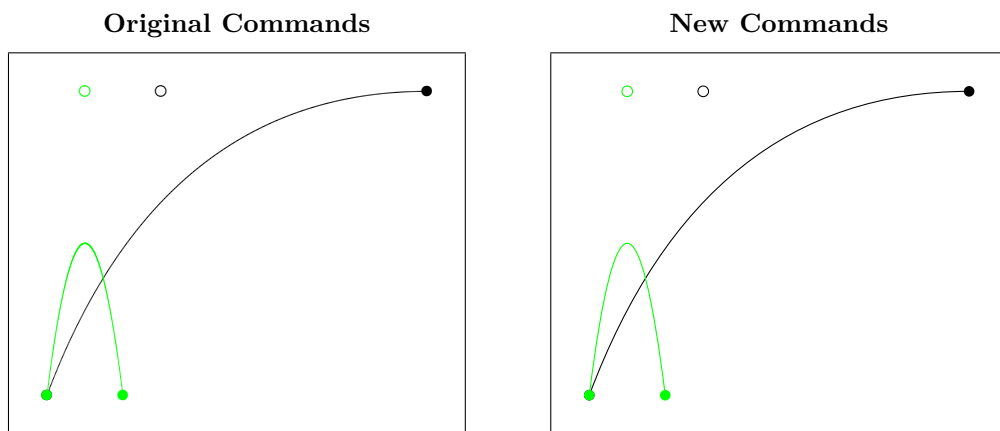


Figure 7: Quadratic Bezier curves

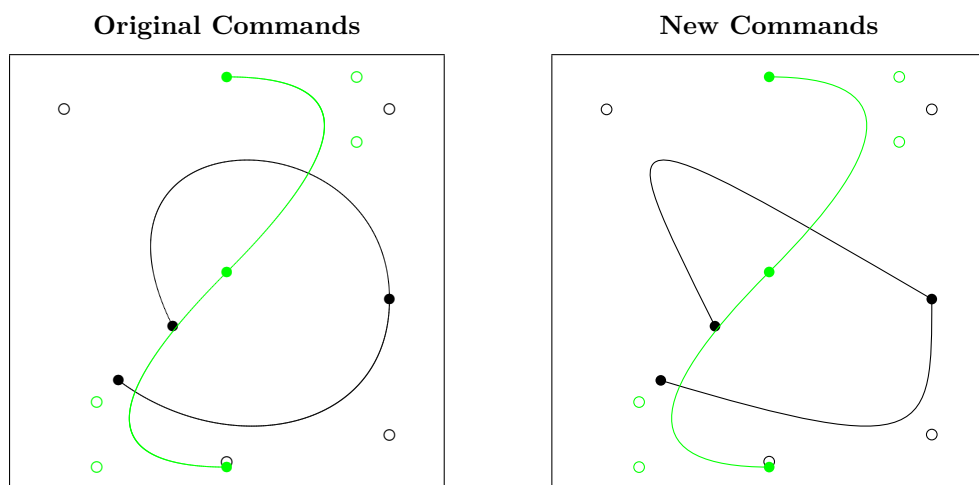


Figure 8: Cubic Bezier curves

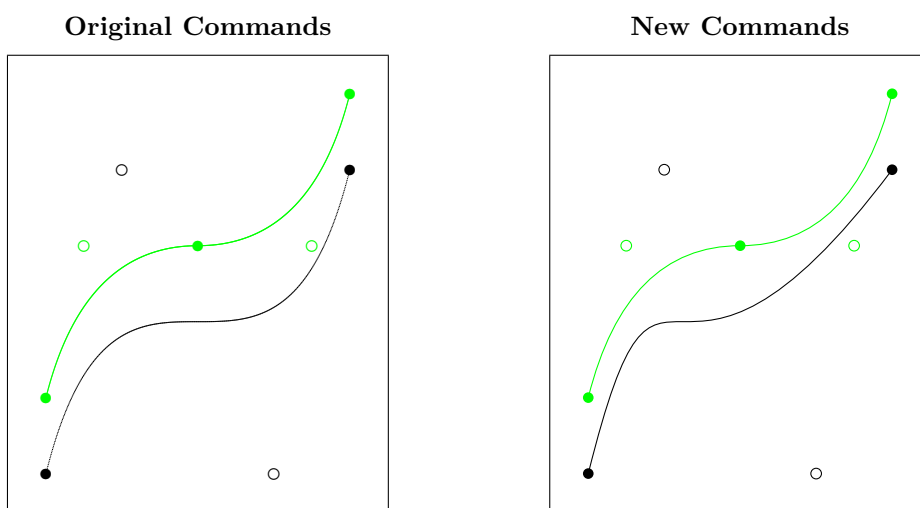


Figure 9: Quadratic (green) and Cubic Bezier curves

### 3 Implementation

Unlike other packages that have reimplemented or extended some of the commands from Standard L<sup>A</sup>T<sub>E</sub>X's `picture` environment, we do not use special fonts, nor draw arbitrary shapes by the means of myriads of small (point) characters, nor do we use sophisticated programming in some back-end programming language.

In its present state, this implementation supports just PostScript and PDF as back-end formats. It just calculates the necessary control points and uses primitive path drawing operators.

```
1 ⟨*package⟩
```

#### 3.1 Initialisation

`\Gin@codes` First we save the catcodes of some characters, and set them to fixed values whilst this file is being read. (This is done in almost the same manner as in the `graphics` and `color` packages. Alas, we don't need nor want to have `*` as part of control sequence names, so we omit it here.)

```
2 \edef\Gin@codes{%
3 \catcode'\noexpand\^^A\the\catcode'\^^A\relax
4 \catcode'\noexpand\" \the\catcode'\\" \relax
5 % \catcode'\noexpand\*\the\catcode'\*\relax
6 \catcode'\noexpand\!\the\catcode'\!\relax
7 \catcode'\noexpand\:\the\catcode'\:\relax}

8 \catcode'\^^A=\catcode'\%
9 \@makeother\"%
10 % \catcode'\*=11
11 \@makeother\!%
12 \@makeother\:%
```

#### 3.2 Preliminaries

`\@defaultunitsset` Command to accept a number or length expression. Added to L<sup>A</sup>T<sub>E</sub>X 2020-10-01 release but provided here for older releases.

Set a length register, `#1`, accepting number or an etex length expression, `#2`, with default unit, `#3`.

`#3` can be a literal unit such as `cm` or a length register such as `\unitlength`.

This is used in all `picture` commands that take picture coordinates. So `\put(2,2)` as previously but now `\put(\textwidth-5cm,0.4\textheight)` Note that you can only use expressions with lengths, `\put(1+2,0)` is not supported.

```
13 \def\@defaultunitsset#1#2#3{%
14 \@defaultunits#1\dimexpr#2#3\relax\relax\@nnil}
```

`\pIIe@mode` The first two of these commands determine how the `pict2e` package works internally; they should be defined properly by the `p2e-⟨driver⟩.def` files. (See file `p2e-drivers.dtx` for details and sample implementations.)

`\pIIe@code`  
`\Gin@driver` The latter command is well known from the `graphics` and `color` packages from the Standard L<sup>A</sup>T<sub>E</sub>X graphics bundle; it should be set by a package option—most likely in a (system dependent) configuration file `pict2e.cfg`. (File `p2e-drivers.dtx` contains an example configuration file suitable for the t<sub>E</sub>X and T<sub>E</sub>Xlive distributions; it will be extracted as `pict2e-example.cfg`.)

```
15 \newcommand*\pIIe@mode{-1}
16 \newcommand*\pIIe@code[1]{}
17 \providecommand*\Gin@driver{}
```

`\pIie@tempa` At times, we need some temporary storage bins. However, we only use some macros and do  
`\pIie@tempb` not allocate any new registers; the “superfluous” ones from the picture module of the kernel  
`\pIie@tempc` (`ltpictur.dtx`) and the general scratch registers should suffice.

```

18 \newcommand*\pIie@tempa{}
19 \newcommand*\pIie@tempb{}
20 \newcommand*\pIie@tempc{}

```

### 3.3 Option processing

The driver options are not much of a surprise: they are similar to those of the `graphics` and `color` packages.

```

21 \DeclareOption{dvips}{\def\Gin@driver{dvips.def}}
22 \DeclareOption{xdvi}{\ExecuteOptions{dvips}}
23 \DeclareOption{dviptdf}{\def\Gin@driver{dviptdf.def}}
24 \DeclareOption{dviptdfm}{\def\Gin@driver{dviptdfm.def}}
25 \DeclareOption{dviptdfmx}{\def\Gin@driver{dviptdfmx.def}}
26 \DeclareOption{pdftex}{\def\Gin@driver{pdftex.def}}
27 \DeclareOption{luatex}{\def\Gin@driver{luatex.def}}
28 \DeclareOption{xetex}{\def\Gin@driver{xetex.def}}
29 \DeclareOption{dvipsone}{\def\Gin@driver{dvipsone.def}}
30 \DeclareOption{dviwindo}{\ExecuteOptions{dvipsone}}
31 \DeclareOption{oztex}{\ExecuteOptions{dvips}}
32 \DeclareOption{textures}{\def\Gin@driver{textures.def}}
33 \DeclareOption{pctexps}{\def\Gin@driver{pctexps.def}}
34 \DeclareOption{pctex32}{\def\Gin@driver{pctex32.def}}
35 \DeclareOption{vtex}{\def\Gin@driver{vtex.def}}

```

Request “original” L<sup>A</sup>T<sub>E</sub>X mode.

```

36 \DeclareOption{original}{\def\pIie@mode{0}}

```

`\ifpIie@pdfliteral@ok` Check, whether if `\pIie@pdfliteral` is given in the driver file or `\pdfliteral` available  
`\pIie@pdfliteral` directly.

```

37 \newif\ifpIie@pdfliteral@ok
38 \pIie@pdfliteral@oktrue
39 \ifx\pIie@pdfliteral\@undefined
40   \ifx\pdfliteral\@undefined
41     \pIie@pdfliteral@okfalse
42     \def\pIie@pdfliteral#1{%
43       \PackageWarning{pict2e}{pdfliteral not supported}%
44     }%
45   \else
46     \let\pIie@pdfliteral\pdfliteral
47   \fi
48 \fi

```

`\pIie@buttcap` Do `\buttcap` only if available.

```

49 \def\pIie@buttcap{%
50   \ifpIie@pdfliteral@ok
51     \buttcap
52   \fi
53 }

```

`\pIie@FAL` Some macros to parametrize the shape of the vector outline. The following values are “hand  
`\pIie@FAW` optimized” with the aim of emulating L<sup>A</sup>T<sub>E</sub>X-style arrows. They also seem suitable for our  
`\pIie@CAW` PSTricks-style arrows. See Figures 10 and 11.

```

\pIie@FAI
54 \newcommand*\pIie@FAL{1.52}%
55 \newcommand*\pIie@FAW{3.2}%
56 \newcommand*\pIie@CAW{1.5pt}%
57 \newcommand*\pIie@FAI{0.25}%

```

`\ltxarrows` The following user-level macros can be used to change the arrow style (L<sup>A</sup>T<sub>E</sub>X-style is the default).

```
\pstarrows
58 \newcommand*\ltxarrows{%
59   \let\pIIE@vector=\pIIE@vector@ltx
60 }
61 \newcommand*\pstarrows{%
62   \let\pIIE@vector=\pIIE@vector@pst
63 }

64 \DeclareOption{ltxarrows}{\AtEndOfPackage{\ltxarrows}}
65 \DeclareOption{pstarrows}{\AtEndOfPackage{\pstarrows}}
```

`\pIIE@debug@comment` This makes debugging easier.

```
66 \newcommand*\pIIE@debug@comment{}
67 \DeclareOption{debug}{%
68   \def\pIIE@debug@comment{^^J^^J\@percentchar\space >>> pict2e <<<^^J}%
69   \begingroup
70     \ifundefined{pdfcompresslevel}{\global\pdfcompresslevel\z@}%
71   \endgroup}
```

A special variant of debugging. (Obsolescent? Once used for performance measurements: arctan vs. pyth-add versions of `\vector`.)

```
72 \DeclareOption{hide}{\AtEndOfPackage{%
73   % \def\pIIE@code#1{}%
74   \let\pIIE@code\@gobble
75 }}
```

Unknown options default to mode “original.”

```
76 \DeclareOption*{\ExecuteOptions{original}}
```

By default, arrows are in the L<sup>A</sup>T<sub>E</sub>X style.

```
77 \ExecuteOptions{ltxarrows}
```

Like the graphics and color packages, we support a configuration file. (See file `p2e-drivers.dtx` for details and an example.)

```
78 \InputIfFileExists{pict2e.cfg}{\}{}%
```

This now should make clear which “mode” and “code” we should use.

```
79 \ProcessOptions\relax
```

### 3.4 Output driver check

```
80 \ifnum\pIIE@mode=\z@
81   \PackageInfo{pict2e}{Package option ‘original’ requested}
82 \else
```

This code fragment is more or less cloned from the graphics and color packages.

```
83   \if!\Gin@driver!
84     \PackageError{pict2e}
85       {No driver specified at all}
86       {You should make a default driver option in a file\MessageBreak
87         pict2e.cfg\MessageBreak eg: \protect\ExecuteOptions{dvips}}%
88   \else
89     \PackageInfo{pict2e}{Driver file: \Gin@driver}
90     \@ifundefined{ver@\Gin@driver}{\input{\Gin@driver}}{}
91     \PackageInfo{pict2e}{Driver file for pict2e: p2e-\Gin@driver}
92     \InputIfFileExists{p2e-\Gin@driver}{\}{}%
93     \PackageError{pict2e}%
94       {Driver file ‘p2e-\Gin@driver’ not found}%
95     {Q: Is the file properly installed? A: No!}}
```

```

96 \fi
97 \fi

```

### 3.5 Mode check

For PostScript and PDF modes.

```

98 \ifnum\pIIE@mode>\z@
99 \ifnum\pIIE@mode<\thr@@
100 \RequirePackage{trig}

```

```

\pIIE@oldline Saved versions of some macros. (Or dummy definitions.)
\pIIE@old@sline 101 \let\pIIE@old@sline\line
\pIIE@old@vector 102 \let\pIIE@old@sline\@sline
\pIIE@old@circle 103 \let\pIIE@old@vector\vector
\pIIE@old@dot 104 \let\pIIE@old@circle\@circle
\pIIE@old@bezier 105 \let\pIIE@old@dot\@dot
\pIIE@old@cbezier 106 \let\pIIE@old@bezier\@bezier
\pIIE@old@oval 107 \AtBeginDocument{%
\pIIE@old@oval 108 \ifundefined{@cbezier}{%
109 \def\pIIE@old@cbezier[#1](#2,#3)(#4,#5)(#6,#7)(#8,#9){}%
110 }\let\pIIE@old@cbezier\@cbezier}}
111 \let\pIIE@oldoval\oval
112 \let\pIIE@old@oval\@oval

```

`\OriginalPictureCmds` Switches back to the original definitions; for testing and demonstration purposes only.

```

113 \newcommand*\OriginalPictureCmds{%
114 \let\@sline\pIIE@old@sline
115 \let\line\pIIE@oldline
116 \let\vector\pIIE@oldvector
117 \let\@circle\pIIE@old@circle
118 \let\@dot\pIIE@old@dot
119 \let\@bezier\pIIE@old@bezier
120 \let\@cbezier\pIIE@old@cbezier
121 \renewcommand*\oval[1][\pIIE@oldoval]%
122 \let\@oval\pIIE@old@oval
123 }

```

Overambitious drivers.

```

124 \else
125 \PackageError{pict2e}
126 {Unsupported mode (\pIIE@mode) specified}
127 {The driver you specified requested a mode\MessageBreak
128 not supported by this version of this package}
129 \fi

```

Incapable drivers.

```

130 \else
131 \ifnum\pIIE@mode<\z@
132 \PackageError{pict2e}
133 {No suitable driver specified}
134 {You should make a default driver option in a file\MessageBreak
135 pict2e.cfg\MessageBreak eg: \protect\ExecuteOptions{dvips}}
136 \fi
137 \fi

```

Big switch, completed near the end of the package (see page 35).

```

138 \ifnum\pIIE@mode>\z@

```



### 3.6 Graphics operators

The following definitions allow the PostScript and PDF operations below to share some of the code.

```
139 \ifcase\pIIE@mode\relax
```

PostScript

```
\pIIE@moveto@op
\pIIE@lineto@op 140 \or
\pIIE@setlinewidth@op 141 \newcommand*\pIIE@moveto@op{moveto}
\pIIE@stroke@op 142 \newcommand*\pIIE@lineto@op{lineto}
\pIIE@fill@op 143 \newcommand*\pIIE@setlinewidth@op{setlinewidth}
\pIIE@curveto@op 144 \newcommand*\pIIE@stroke@op{stroke}
\pIIE@concat@op 145 \newcommand*\pIIE@fill@op{fill}
\pIIE@closepath@op 146 \newcommand*\pIIE@curveto@op{curveto}
147 \newcommand*\pIIE@concat@op{concat}
148 \newcommand*\pIIE@closepath@op{closepath}
```

PDF

```
\pIIE@moveto@op
\pIIE@lineto@op 149 \or
\pIIE@setlinewidth@op 150 \newcommand*\pIIE@moveto@op{m}
\pIIE@stroke@op 151 \newcommand*\pIIE@lineto@op{l}
\pIIE@fill@op 152 \newcommand*\pIIE@setlinewidth@op{w}
\pIIE@curveto@op 153 \newcommand*\pIIE@stroke@op{S}
\pIIE@concat@op 154 \newcommand*\pIIE@fill@op{f}
\pIIE@closepath@op 155 \newcommand*\pIIE@curveto@op{c}
156 \newcommand*\pIIE@concat@op{cm}
157 \newcommand*\pIIE@closepath@op{h}
```

(Currently, there are no other modes.)

```
158 \fi
```

### 3.7 Low-level operations

#### 3.7.1 Collecting the graphics instructions and handling the output

\pIIE@GRAPH We collect all PostScript/PDF output code for a single picture object in a token register.

```
\pIIE@addtoGraph 159 \ifdefinable\pIIE@GRAPH{\newtoks\pIIE@GRAPH}
160 \newcommand*\pIIE@addtoGraph[1]{%
161 \begingroup
162 \edef\x{\the\pIIE@GRAPH\space#1}%
163 \global\pIIE@GRAPH\expandafter\x}%
164 \endgroup}
```

\pIIE@fillGraph The path will either be filled ...

```
165 \newcommand*\pIIE@fillGraph{\begingroup \@tempswatrue\pIIE@drawGraph}
```

\pIIE@strokeGraph ... or stroked.

```
166 \newcommand*\pIIE@strokeGraph{\begingroup \@tempswafalse\pIIE@drawGraph}
```

\pIIE@drawGraph Common code. When we are done with collecting the path of the picture object, we output the contents of the token register.

```
167 \newcommand*\pIIE@drawGraph{%
168 \edef\x{\pIIE@debug@comment\space
```

Instead of scaling individual coordinates, we scale the graph as a whole (pt→bp); see Section 3.8.1.

```
169 \pIIE@scale@PTtoBP}%
170 \if@tempswa
```

```

171     \edef\y{\pIIE@fill@op}%
172     \else
173     \edef\x{\x\space
174     \strip@pt\@wholewidth\space\pIIE@setlinewidth@op
175     \pIIE@linecap\pIIE@linejoin\space}%
176     \edef\y{\pIIE@stroke@op}%
177     \fi
178     \expandafter\pIIE@code\expandafter{%
179     \expandafter\x\the\pIIE@GRAPH\space\y}%
Clear the graph and the current point after output.
180     \global\pIIE@GRAPH{}\xdef\pIIE@CPx{}\xdef\pIIE@CPy{}%
181     \endgroup}

```

### 3.7.2 Auxilliary macros

The following macros save us a plethora of tokens in subsequent code.

Note that since we are using `\@tempdima` and `\@tempdimb` both here and in medium-level macros below, we must be careful not to spoil their values.

`\pIIE@CPx` The lengths (coordinates) given as arguments will be stored as “real” numbers using the common trick; i.e., they are put in ‘dimen’ registers, scaled by  $2^{16}$ . At the same time, we  
`\pIIE@CPy` remember the “current point.” (Not strictly necessary for PostScript, but for some operations  
`\pIIE@add@CP` in PDF, e.g., *rcurveto* emulation.)

```

182 \newcommand*\pIIE@CPx{} \newcommand*\pIIE@CPy{}
183 \newcommand*\pIIE@add@CP[2]{%
184   \begingroup
185   \@tempdima#1\xdef\pIIE@CPx{\the\@tempdima}%
186   \@tempdimb#2\xdef\pIIE@CPy{\the\@tempdimb}%
187   \pIIE@addtoGraph{\strip@pt\@tempdima\space\strip@pt\@tempdimb}%
188   \endgroup}

```

`\pIIE@add@nums` Similar, but does not set the “current point.” Values need not be coordinates (e.g., may be scaling factors, etc.).

```

189 \newcommand*\pIIE@add@nums[2]{%
190   \begingroup
191   \@tempdima#1\relax
192   \@tempdimb#2\relax
193   \pIIE@addtoGraph{\strip@pt\@tempdima\space\strip@pt\@tempdimb}%
194   \endgroup}

```

`\pIIE@add@num` Likewise, for a single argument.

```

195 \newcommand*\pIIE@add@num[1]{%
196   \begingroup
197   \@tempdima#1\relax
198   \pIIE@addtoGraph{\strip@pt\@tempdima}%
199   \endgroup}

```

## 3.8 Medium-level operations

### 3.8.1 Transformations

Transformation operators; not all are currently used. (Hence, some are untested.)

`\pIIE@PTtoBP` Scaling factor, used below. “pt→bp” ( $72/72.27 \approx 0.99626401$ ). Note the trailing space! (Don’t delete it, it saves us some tokens.)

```

200 \newcommand*\pIIE@PTtoBP{0.99626401 }
201 \ifcase\pIIE@mode\relax

```

```

\pIIE@concat PostScript: Use some operators directly.
\pIIE@translate 202 \or
\pIIE@rotate 203 \newcommand*\pIIE@concat[6]{%
\pIIE@scale 204 \begin{group}
\pIIE@scale@PTtoBP 205 \pIIE@addtoGraph{[%
206 \@tempdima#1\relax \@tempdimb#2\relax
207 \pIIE@add@nums\@tempdima\@tempdimb
208 \@tempdima#3\relax \@tempdimb#4\relax
209 \pIIE@add@nums\@tempdima\@tempdimb
210 \@tempdima#5\relax \@tempdimb#6\relax
211 \pIIE@add@nums\@tempdima\@tempdimb
212 \pIIE@addtoGraph[] \pIIE@concat@op}%
213 \end{group}}
214 \newcommand*\pIIE@translate[2]{\pIIE@add@nums{#1}{#2}\pIIE@addtoGraph{translate}}
215 \newcommand*\pIIE@rotate[1]{\pIIE@add@num{#1}\pIIE@addtoGraph{rotate}}
216 \newcommand*\pIIE@scale[2]{\pIIE@add@nums{#1}{#2}\pIIE@addtoGraph{scale}}
217 \newcommand*\pIIE@scale@PTtoBP{\pIIE@PTtoBP \pIIE@PTtoBP scale}

```

```

\pIIE@concat PDF: Emulate. :- (
\pIIE@translate 218 \or
\pIIE@rotate 219 \newcommand*\pIIE@concat[6]{%
\pIIE@scale 220 \begin{group}
\pIIE@scale@PTtoBP 221 \@tempdima#1\relax \@tempdimb#2\relax
222 \pIIE@add@nums\@tempdima\@tempdimb
223 \@tempdima#3\relax \@tempdimb#4\relax
224 \pIIE@add@nums\@tempdima\@tempdimb
225 \@tempdima#5\relax \@tempdimb#6\relax
226 \pIIE@add@nums\@tempdima\@tempdimb
227 \pIIE@addtoGraph\pIIE@concat@op
228 \end{group}}
229 \newcommand*\pIIE@translate[2]{\pIIE@concat\p@z@\z@\p@{#1}{#2}}
230 \newcommand*\pIIE@rotate[1]{%
231 \begin{group}
232 \@tempdima#1\relax
233 \edef\pIIE@tempa{\strip@pt\@tempdima}%
234 \CalculateSin\pIIE@tempa
235 \CalculateCos\pIIE@tempa
236 \edef\pIIE@tempb{\UseSin\pIIE@tempa}%
237 \edef\pIIE@tempc{\UseCos\pIIE@tempa}%
238 \pIIE@concat{\pIIE@tempc\p@}{\pIIE@tempb\p@}%
239 {-\pIIE@tempb\p@}{\pIIE@tempc\p@}\z@\z@
240 \end{group}}
241 \newcommand*\pIIE@scale[2]{\pIIE@concat{#1}\z@\z@{#2}\z@\z@}
242 \newcommand*\pIIE@scale@PTtoBP{\pIIE@PTtoBP 0 0 \pIIE@PTtoBP 0 0 \pIIE@concat@op}

```

(Currently, there are no other modes.)

```
243 \fi
```

### 3.8.2 Path definitions

\pIIE@moveto Simple things ...

```

244 \newcommand*\pIIE@moveto[2]{%
245 \pIIE@add@CP{#1}{#2}\pIIE@addtoGraph\pIIE@moveto@op}

```

\pIIE@lineto ... have to be defined, too.

```

246 \newcommand*\pIIE@lineto[2]{%
247 \pIIE@add@CP{#1}{#2}\pIIE@addtoGraph\pIIE@lineto@op}

```

We'll use `\pIe@rcurveto` to draw quarter circles. (`\circle` and `\oval`).

```
248 \ifcase\pIe@mode\relax
```

`\pIe@rcurveto` PostScript: Use the “rcurveto” operator directly.

```
249 \or
250 \newcommand*\pIe@rcurveto[6]{%
251 \begingroup
252 \@tempdima#1\relax \@tempdimb#2\relax
253 \pIe@add@nums\@tempdima\@tempdimb
254 \@tempdima#3\relax \@tempdimb#4\relax
255 \pIe@add@nums\@tempdima\@tempdimb
256 \@tempdima#5\relax \@tempdimb#6\relax
257 \pIe@add@CP\@tempdima\@tempdimb
258 \pIe@addtoGraph{rcurveto}%
259 \endgroup}
```

`\pIe@rcurveto` PDF: It's necessary to emulate the PostScript operator “rcurveto”. For this, the “current point” must be known, i.e., all macros which change the “current point” must set `\pIe@CPx` and `\pIe@CPy`.

```
260 \or
261 \newcommand*\pIe@rcurveto[6]{%
262 \begingroup
263 \@tempdima#1\advance\@tempdima\pIe@CPx\relax
264 \@tempdimb#2\advance\@tempdimb\pIe@CPy\relax
265 \pIe@add@nums\@tempdima\@tempdimb
266 \@tempdima#3\advance\@tempdima\pIe@CPx\relax
267 \@tempdimb#4\advance\@tempdimb\pIe@CPy\relax
268 \pIe@add@nums\@tempdima\@tempdimb
269 \@tempdima#5\advance\@tempdima\pIe@CPx\relax
270 \@tempdimb#6\advance\@tempdimb\pIe@CPy\relax
271 \pIe@add@CP\@tempdima\@tempdimb
272 \pIe@addtoGraph\pIe@curveto@op
273 \endgroup}
```

(Currently, there are no other modes.)

```
274 \fi
```

`\pIe@curveto` This is currently only used for Bezier curves and for drawing the heads of L<sup>A</sup>T<sub>E</sub>X-like arrows. Note: It's the same for PostScript and PDF.

```
275 \newcommand*\pIe@curveto[6]{%
276 \begingroup
277 \@tempdima#1\relax \@tempdimb#2\relax
278 \pIe@add@nums\@tempdima\@tempdimb
279 \@tempdima#3\relax \@tempdimb#4\relax
280 \pIe@add@nums\@tempdima\@tempdimb
281 \@tempdima#5\relax \@tempdimb#6\relax
282 \pIe@add@CP\@tempdima\@tempdimb
283 \pIe@addtoGraph\pIe@curveto@op
284 \endgroup}
```

`\pIe@closepath`

```
285 \newcommand*\pIe@closepath{\pIe@addtoGraph\pIe@closepath@op}
```

### 3.9 “Pythagorean Addition” and Division

`\pIe@pyth` This algorithm is copied from the P<sub>l</sub>CT<sub>E</sub>X package [4] by Michael Wichura, with his permission. Here is his description:

Suppose  $x > 0, y > 0$ . Put  $s = x + y$ . Let  $z = (x^2 + y^2)^{1/2}$ . Then  $z = s \times f$ , where

$$f = (t^2 + (1 - t)^2)^{1/2} = ((1 + \tau^2)/2)^{1/2}$$

and  $t = x/s$  and  $\tau = 2(t - 1/2)$ .

```

286 \newcommand*\pIIE@pyth[3]{%
287 \begingroup
288 \@tempdima=#1\relax
\@tempdima = abs(x)
289 \ifnum\@tempdima<\z@\@tempdima=-\@tempdima\fi
290 \@tempdimb=#2\relax
\@tempdimb = abs(y)
291 \ifnum\@tempdimb<\z@\@tempdimb=-\@tempdimb\fi
\@tempdimb = s = abs(x) + abs(y)
292 \advance\@tempdimb\@tempdima
293 \ifnum\@tempdimb=\z@
\@tempdimc = z = \sqrt{x^2 + y^2}
294 \@tempdimc=\z@
295 \else
\@tempdima = 8 \times abs(x)
296 \multiply\@tempdima 8\relax
\@tempdimc = 8 t = 8 \times abs(x)/s
297 \pIIE@divide\@tempdima\@tempdimb\@tempdimc
\@tempdimc = 4\tau = (8 t - 4)
298 \advance\@tempdimc -4pt
299 \multiply\@tempdimc 2
300 \edef\pIIE@tempa{\strip@pt\@tempdimc}%
\@tempdima = (8 \tau)^2
301 \@tempdima=\pIIE@tempa\@tempdimc
\@tempdima = [64 + (8 \tau)^2]/2 = (8 f)^2
302 \advance\@tempdima 64pt
303 \divide\@tempdima 2\relax
initial guess at \sqrt{u}
304 \@dashdim=7pt
\@dashdim = \sqrt{u}
305 \pIIE@@pyth\pIIE@@pyth\pIIE@@pyth
306 \edef\pIIE@tempa{\strip@pt\@dashdim}%
307 \@tempdimc=\pIIE@tempa\@tempdimb
\@tempdimc = z = (8 f) \times s/8
308 \global\divide\@tempdimc 8
309 \fi
310 \edef\x{\endgroup#3=\the\@tempdimc}%
311 \x}

\pIIE@@pyth \@dashdim = g \leftarrow (g + u/g)/2
312 \newcommand*\pIIE@@pyth{%
313 \pIIE@divide\@tempdima\@dashdim\@tempdimc
314 \advance\@dashdim\@tempdimc
315 \divide\@dashdim\tw@}

```

`\pIe@divide` The following macro for division is a slight modification of the macro from `curve2e` by Claudio Beccari with his permission. Real numbers are represented as `dimens` in `pt`.

```
316 \newcommand*\pIe@divide[3]{%
```

All definitions inside a group.

```
317 \begingroup
318 \dimendef\Numer=254\relax \dimendef\Denom=252\relax
319 \countdef\Num=254\relax \countdef\Den=252\relax
320 \countdef\I=250\relax \countdef\Numb=248\relax
321 \Numer #1\relax \Denom #2\relax
```

Make numerator and denominator nonnegative, save sign.

```
322 \ifdim\Denom<\z@ \Denom -\Denom \Numer=-\Numer \fi
323 \ifdim\Numer<\z@ \def\sign{-}\Numer=-\Numer \else \def\sign{}\fi
```

Use `\maxdimen` for  $x/0$  (this should not appear).

```
324 \ifdim\Denom=\z@
325 \edef\Q{\strip@pt\maxdimen}%
326 \PackageWarning{pict2e}%
327 {Division by 0, \sign\strip@pt\maxdimen\space used}{}%
328 \else
```

Converse to integers and find integer part of the ratio. If it is too large (dimension overflow), use `\maxdimen` otherwise find the remainder and start the iteration process to find 6 digits of the decimal expression.

```
329 \Num=\Numer \Den=\Denom
330 \Numb=\Num \divide\Numb\Den
331 \ifnum\Numb>16383
332 \edef\Q{\strip@pt\maxdimen}%
333 \PackageWarning{pict2e}%
334 {Division overflow, \sign\strip@pt\maxdimen\space used}{}%
335 \else
336 \edef\Q{\number\Numb.}%
337 \multiply \Numb\Den \advance\Num -\Numb
338 \I=6\relax
339 \@whilenum \I>\z@ \do{\pIe@@divide\advance\I\m@ne}%
340 \fi
341 \fi
```

A useful trick to define `#3` outside the group without using `\global` (if the macro is used inside another group.)

```
342 \edef\tempend{\noexpand\endgroup\noexpand#3=\sign\Q\p@}%
343 \tempend}
```

`\pIe@@divide` Iteration macro for finding decimal expression of the ratio. `\Num` is the remainder of the previous division, `\Den` is the denominator (both are integers).

```
344 \def\pIe@@divide{%
```

Reduce both numerator and denominator if necessary to avoid overflow in the next step.

```
345 \@whilenum \Num>214748364 \do{\divide\Num\tw@ \divide\Den\tw@}%
```

Find the next digit of the decimal expression.

```
346 \multiply \Num 10
347 \Numb=\Num \divide\Numb\Den
348 \edef\Q{\Q\number\Numb}%
```

Find the remainder.

```
349 \multiply \Numb\Den \advance \Num -\Numb
```

Stop the iteration if the remainder is zero.

```
350 \ifnum\Num>\z@\else\I=0\fi}
```

### 3.10 High-level operations

`\pIe@checkslopeargs` Common code for `\line` and `\vector`.

```

351 \newcommand*\pIe@checkslopeargsline[2]{%
352   \pIe@checkslopeargs{#1}{#2}{16383}}
353 \newcommand*\pIe@checkslopeargsvector[2]{%
354   \pIe@checkslopeargs{#1}{#2}{1000}}
355 \newcommand*\pIe@checkslopeargs[3]{%
356   \edef\@tempa{#1}\expandafter\pIe@checkslopearg\@tempa.:{#3}%
357   \edef\@tempa{#2}\expandafter\pIe@checkslopearg\@tempa.:{#3}%

```

A bit incompatible with Standard L<sup>A</sup>T<sub>E</sub>X: slope (0,0) raises an error.

```

358   \ifdim #1p@=\z@ \ifdim #2p@=\z@ \@badlinearg \fi\fi}
359 \def\pIe@checkslopearg #1.#2:#3{%
360   \def\@tempa{#1}%
361   \ifx\@tempa\empty\def\@tempa{0}\fi
362   \ifx\@tempa\space\def\@tempa{0}\fi
363   \ifnum\ifnum\@tempa<\z@-\fi\@tempa>#3 \@badlinearg \fi}
364 \def\@badlinearg{\PackageError
365   {pict2e}{Bad \protect\line\space or \protect\vector\space argument}{}}

```

#### 3.10.1 Line

`\line` `\line( $\langle x,y \rangle$ ){ $\langle l_x \rangle$ }`:

```

366 \def\line(#1,#2)#3{%
367   \begingroup
368   \pIe@checkslopeargsline{#1}{#2}%
369   \@tempdima=#1pt\relax \@tempdimb=#2pt\relax
370   \@defaultunitsset\@linelen{#3}\unitlength
371   \ifdim\@linelen<\z@ \@badlinearg \else
372     \pIe@sline
373     \pIe@moveto\z@\z@
374     \pIe@lineto\@xdim\@ydim
375     \pIe@strokeGraph

```

Simulated bounding box

```

376     \box\@tempboxa
377     \fi
378   \endgroup}

```

`\pIe@sline` Common code for `\line` and `\vector`.

```

379 \newcommand*\pIe@sline{%

```

Calculation of the endpoints `\@xdim`, `\@ydim` (used for `\line` only).

```

380   \ifdim\@tempdima=\z@
381     \ifdim\@tempdimb<\z@\@linelen-\@linelen\fi
382     \@ydim=\@linelen
383     \@xdim=\z@
384   \else
385     \ifdim\@tempdima<\z@\@linelen-\@linelen\fi
386     \ifdim\@tempdimb=\z@
387       \@xdim=\@linelen
388       \@ydim=\z@
389     \else
390       \pIe@divide\@tempdimb\@tempdima\dimen@
391       \@ydim=\strip@pt\dimen@\@linelen
392       \@xdim=\@linelen
393     \fi
394   \fi

```

Prepare a box that can be used as a bounding box for `\line` and `\vector` to achieve the same behavior as standard L<sup>A</sup>T<sub>E</sub>X outside of a picture environment.

```

395 \ovxx=\ifnum\@xdim=\z@ \z@\else\@linelen\fi
396 \ovyy=\ifnum\@ydim<\z@ \z@\else\@ydim\fi
397 \ovdy=\ifnum\@ydim<\z@ -\@ydim\else\z@\fi
398 \setbox\@tempboxa\hbox{%
399 \vrule\@height \ovyy \@depth \ovdy \@width \z@
400 \vrule\@height \z@ \@depth \z@ \@width \ovxx}}

```

### 3.10.2 Vector

`\vector` Unlike `\line`, `\vector` must be redefined, because the kernel version checks for illegal slope arguments.

`\vector(\langle x,y\rangle)\{\langle l_x\rangle\}`: Instead of calculating  $\theta = \arctan \frac{y}{x}$ , we use “pythagorean addition” [4] to determine  $s = \sqrt{x^2 + y^2}$  and to obtain the length of the vector  $l = l_x \cdot \frac{s}{x}$  and the values of  $\sin \theta = \frac{y}{s}$  and  $\cos \theta = \frac{x}{s}$  for the rotation of the coordinate system.

```

401 \def\vector(#1,#2)#3{%
402 \begingroup
403 \pIIE@checkslopeargsvector{#1}{#2}%
404 \@tempdima=#1pt\relax \@tempdimb=#2pt\relax
405 \@defaultunitsset\@linelen{#3}\unitlength
406 \ifdim\@linelen<\z@ \@badlinearg \else
407 \pIIE@sline
408 \@defaultunitsset\@linelen{#3}\unitlength
409 \pIIE@pyth{\@tempdima}{\@tempdimb}\dimen@
410 \ifdim\@tempdima=\z@ \else
411 \ifdim\@tempdimb=\z@ \else

```

This calculation is only necessary, if the vector is actually sloped.

```

412 \pIIE@divide\dimen@{\@tempdima}\@xdim
413 \@linelen\strip@pt\@xdim\@linelen
414 \ifdim\@linelen<\z@\@linelen-\@linelen\fi
415 \fi
416 \fi

```

$\sin \theta$  and  $\cos \theta$

```

417 \pIIE@divide{\@tempdimb}\dimen@\@ydim
418 \pIIE@divide{\@tempdima}\dimen@\@xdim

```

Rotate the following vector/arrow outlines by angle  $\theta$ :

$\cos \theta \quad \sin \theta \quad -\sin \theta \quad \cos \theta \quad 0 \quad 0$

```

419 \pIIE@concat\@xdim\@ydim{-\@ydim}\@xdim\z@\z@

```

Internal command to draw the outline of the vector/arrow.

```

420 \pIIE@vector
421 \pIIE@fillGraph

```

Simulated bounding box

```

422 \box\@tempboxa
423 \fi
424 \endgroup}

```

`\pIIE@vector` This command should be `\def`’ed or `\let` to a macro that generates the vector’s outline path. Now initialized by package options, via `\AtEndOfPackage`.

```

425 \newcommand*\pIIE@vector{}

```



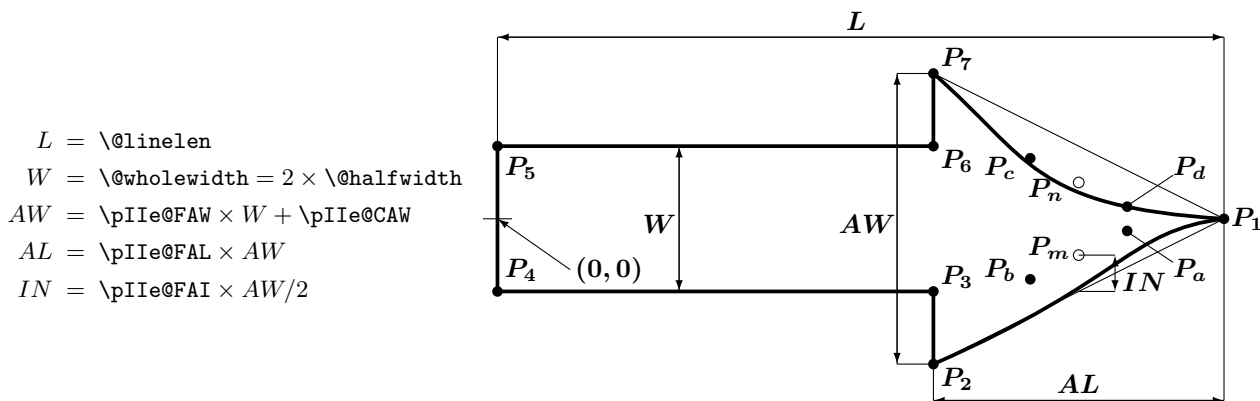
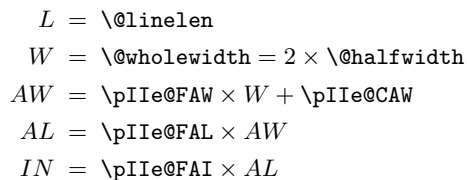


Figure 10: Sketch of the path drawn by the L<sup>A</sup>T<sub>E</sub>X-like implementation of `\vector`. (Note: We are using the redefined macros of `pict2e`!)

`\pIIe@vector@ltx` The arrow outline. (Not yet quite the same as with L<sup>A</sup>T<sub>E</sub>X's fonts.)

Note that only the arrow head will be drawn, if the length argument of the `\vector` command is smaller than the calculated length of the arrow head.



444  $P_6$   
445  $\backslash\mathrm{pIIE@lineto}\@xdim\{\@halfwidth\}\%$   
 $\backslash\mathrm{fi}$   
 $P_7$   
446  $\backslash\mathrm{pIIE@lineto}\@xdim\@ydim$   
 $P_c \quad P_d \quad P_1$   
447  $\backslash\mathrm{pIIE@curveto}\@cldnwd\@cldnht\@ovro\@ovri\@linelen\z@}$

`\pIIe@vector@p` The arrow outline. Note that only the arrowhead will be drawn, if the length argument of the `\vector` command is smaller than the calculated length of the arrow head.

```

448 \newcommand*\pIIE@vector@pst{%
449   \@ydim\pIIE@FAW\@wholewidth \advance\@ydim\pIIE@CAW\relax
450   \@ovxx\pIIE@FAL\@ydim
451   \@xdim\@linelen \advance\@xdim-\@ovxx
452   \divide\@ydim\tw@
453   \@ovy\@ydim \advance\@ovy-\@halfwidth
454   \@ovdx\pIIE@FAI\@ovxx
455   \pIIE@divide\@ovdx\@ydim\@tempdimc
456   \@ovxx\strip@pt\@ovy\@tempdimc
457   \advance\@ovxx\@xdim
458   \advance\@ovdx\@xdim

```

$$\begin{aligned}
&P_1 \\
&\backslash \mathrm{pI} \mathrm{I} \mathrm{e} @ \mathrm{moveto} \backslash @ \mathrm{linelen} \backslash \mathrm{z} @ \\
&P_2 \\
&\backslash \mathrm{pI} \mathrm{I} \mathrm{e} @ \mathrm{lineto} \backslash @ \mathrm{xdim} \{ - \backslash @ \mathrm{ydim} \} \% \\
&\backslash \mathrm{idim} \backslash @ \mathrm{xdim} > \backslash \mathrm{z} @ \\
&P_3 \\
&\backslash \mathrm{pI} \mathrm{I} \mathrm{e} @ \mathrm{lineto} \backslash @ \mathrm{ovxx} \{ - \backslash @ \mathrm{halfwidth} \} \% \\
&P_4 \\
&\backslash \mathrm{pI} \mathrm{I} \mathrm{e} @ \mathrm{lineto} \backslash \mathrm{z} @ \{ - \backslash @ \mathrm{halfwidth} \} \% \\
&P_5 \\
&\backslash \mathrm{pI} \mathrm{I} \mathrm{e} @ \mathrm{lineto} \backslash \mathrm{z} @ \{ \backslash @ \mathrm{halfwidth} \} \%
\end{aligned}$$

```


$$P_6$$

465 \pIIE@lineto\@ovxx{\@halfwidth}%
466 \else

$$P_i$$

467 \pIIE@lineto\@ovdx\z@
468 \fi

$$P_7$$

469 \pIIE@lineto\@xdim\@ydim

$$P_1$$

470 \pIIE@lineto\@linelen\z@

```

### 3.10.3 Circle and Dot

`\@circle` The circle will either be stroked ...

```
471 \def\@circle#1{\begingroup \@tempswafalse\pIIE@circ{#1}}
```

`\@dot` ... or filled.

```
472 \def\@dot#1{\begingroup \@tempwatrue\pIIE@circ{#1}}
```

`\pIIE@circ` Common code.

```
473 \newcommand*\pIIE@circ[1]{%
```

We need the radius instead of the diameter. Unlike Standard L<sup>A</sup>T<sub>E</sub>X, we check for negative or zero diameter argument.

```
474 \@defaultunitsset\@tempdima{#1}\unitlength
475 \ifdim\@tempdima<\z@ \pIIE@badcircarg \fi
476 \divide\@tempdima\tw@
477 \pIIE@circle\@tempdima
```

With the current state of affairs, we could use `\pIIE@drawGraph` directly; but that would possibly be a case of premature optimisation. (Note to ourselves: Use of the `@tempswa` switch both here and inside quarter-circle! Hence a group is necessary there.)

```
478 \if@tempswa \pIIE@fillGraph \else \buttcap \pIIE@strokeGraph \fi
479 \endgroup}
```

`\pIIE@circle` Approximate a full circle by four quarter circles, use the standard shape of ends.

```
480 \newcommand*\pIIE@circle[1]{%
481 \pIIE@qcircle[1]\z@{#1}\pIIE@qcircle \@ne{#1}%
482 \pIIE@qcircle \tw@{#1}\pIIE@qcircle\thr@@{#1}\pIIE@closepath}
```

`\pIIE@qcircle` Approximate a quarter circle, using cubic Bezier splines.

#1=Switch (0=no ‘moveto’, 1=‘moveto’), #2=Quadrant No., #3=Radius.

0 = 1st Quadrant (NE) 1 = 2nd Quadrant (NW)

2 = 3rd Quadrant (SW) 3 = 4th Quadrant (SE)

(PostScript: We could use the `arc` operator!)

0.55228474983 = “magic number” (see [3]).

Sacrifice a save level (otherwise a private “switch” macro were necessary!)

```
483 \newcommand*\pIIE@qcircle[3][0]{%
484 \begingroup
485 \@ovro#3\relax \@ovri0.55228474983\@ovro
486 \@tempdimc\@ovri \advance\@tempdimc-\@ovro
487 \ifnum#1>\z@ \@tempwatrue \else \@tempswafalse \fi
488 \ifcase#2\relax
NE
489 \pIIE@@qcircle\@ovro\z@\z@\@ovri\@tempdimc\@ovro{-\@ovro}\@ovro
490 \or
```

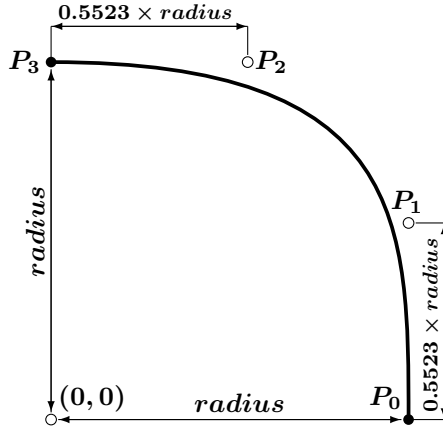


Figure 12: Sketch of the quarter circle path drawn by `\pIIE@qcircle` (NE quarter)

```

NW
491     \pIIE@qcircle\z@\@ovro{-\@ovri}\z@{-\@ovro}\@tempdimc{-\@ovro}{-\@ovro}%
492     \or
SW
493     \pIIE@qcircle{-\@ovro}\z@\z@{-\@ovri}{-\@tempdimc}{-\@ovro}\@ovro{-\@ovro}%
494     \or
SE
495     \pIIE@qcircle\z@{-\@ovro}\@ovri\z@\@ovro{-\@tempdimc}\@ovro\@ovro
496     \fi
497     \endgroup

```

`\pIIE@qcircle` Ancillary macro; saves us some tokens above.

Note: Use of `rcurveto` instead of `curveto` makes it possible (or at least much easier) to re-use this macro for the rounded corners of ovals.

```

498     \newcommand*\pIIE@qcircle[8]{%
499     \if@tempswa\pIIE@moveto{#1}{#2}\fi \pIIE@rcurveto{#3}{#4}{#5}{#6}{#7}{#8}}

```

`\pIIE@badcircarg` Obvious cousin to `\@badlinearg` from the  $\text{\LaTeX}$  kernel.

```

500     \newcommand*\pIIE@badcircarg{%
501     \PackageError{pict2e}%
502     {Illegal argument in \protect\circle(*), \protect\oval, \protect\arc(*) or
503     \protect\circlearc.}%
504     {The radius of a circle, dot, arc or oval corner must be greater than zero.}}%

```

### 3.10.4 Oval

`\maxovalrad` User level command, may be redefined by `\renewcommand*`. It may be given as an explicit (rigid) length (i.e., with unit) or as a number. In the latter case it is used as a factor to be multiplied by `\unitlength`. (`dimen` and `count` registers should work, too.) The default value is 20 pt as specified for the `[<rad>]` argument of `\oval` by the  $\text{\LaTeX}$  manual [1, p. 223].

```

505     \newcommand*\maxovalrad{20pt}

```

`\pIIE@defaultUL` The aforementioned behaviour seems necessary, since [1, p. 223] does not specify explicitly whether the `[<rad>]` argument should be given in terms of `\unitlength` or as an absolute length. This is now re-implemented in terms of `\defaultunitsset`.

`\pIIE@def@UL`

```

506     \newcommand*\pIIE@defaultUL[2]{%
507     \@defaultunitsset\dimen@{#2}\unitlength
508     \edef#1{\the\dimen@}}

```

Hence, we could/should omit the unnecessary argument!?)

```

509 \newcommand*\pIIE@def@UL{}
510 \def\pIIE@def@UL#1\relax#2#3{%
511 % \if!#1!%
512 % \def#2{#3}% \edef ?
513 % \else
514 % \edef#2{\strip@pt\dimen@}%
515 % \fi
516 \edef#2{\the\dimen@}}

```

`\oval` The variant of `\oval` defined here takes an additional optional argument, which specifies the maximum radius of the rounded corners (default = 20pt, as given above). Unlike Standard `\pIIE@maxovalrad`  $\LaTeX$ , we check for negative or zero radius argument. `\pIIE@maxovalrad` is the internal variant of `\maxovalrad`.

```

517 \newcommand*\pIIE@maxovalrad{}
518 \newcommand*\pIIE@oval{}
519 \def\pIIE@oval#1(#2,#3){\ifnextchar[{\@oval(#2,#3)}{\@oval(#2,#3)[]}}
520 \renewcommand*\oval[1][\maxovalrad]{%
521 \begingroup \pIIE@defaultUL\pIIE@maxovalrad{#1}%
522 \ifdim\pIIE@maxovalrad<\z@ \pIIE@badcircarg \fi

```

Can't close the group here, since arguments must be parsed.

```

523 \pIIE@oval}

```

`\@oval` (This is called in turn by the saved original.)

```

524 \def\@oval(#1,#2)[#3]{%

```

In analogy to circles, we need only half of the size value.

```

525 \@defaultunitsset\@ovxx{#1}\unitlength \divide\@ovxx\tw@
526 \@defaultunitsset\@ovyy{#2}\unitlength \divide\@ovyy\tw@
527 \@tempdimc \ifdim\@ovyy>\@ovxx \@ovxx \else \@ovyy \fi
528 \ifdim\pIIE@maxovalrad<\@tempdimc \@tempdimc \pIIE@maxovalrad\relax \fi

```

Subtract the radius of the corners to get coordinates for the straight line segments.

```

529 \@xdim\@ovxx \advance\@xdim-\@tempdimc
530 \@ydim\@ovyy \advance\@ydim-\@tempdimc

```

Determine which parts of the oval we have to draw.

```

531 \pIIE@get@quadrants{#3}%

```

For the whole oval remove use the standard shape of ends.

```

532 \ifnum15=\@tempcnta \pIIE@buttcap \fi

```

“`@tempswa = false`” means, that we have to suppress the ‘moveto’ in the following quadrant.

```

533 \@tempswatrue

```

The following isn't strictly necessary, but yields a single (unfragmented) path even for `[r]` (right half of oval only). Useful for future extensions.

Bits 3 and 0 set? (SE/NE)

```

534 \ifnum9=\@tempcnta
535 \pIIE@qoval\z@{-\@ovyy}{\@xdim}{-\@ovyy}\thr@@\@tempdimc\@ovxx\z@

```

Bit 0 set! (NE)

```

536 \@tempcnta\@ne
537 \fi

```

Bit 0 set? (NE)

```

538 \pIIE@qoval\@ovxx\z@\@ovxx\@ydim\z@\@tempdimc\z@\@ovyy

```

Bit 1 set? (NW)

```

539 \pIIE@qoval\z@\@ovyy{-\@xdim}\@ovyy\@ne\@tempdimc{-\@ovxx}\z@

```

Bit 2 set? (SW)

```
540 \pIIE@qoval{-\@ovxx}\z@{-\@ovxx}{-\@ydim}\tw@\@tempdimc\z@{-\@ovyy}%
```

Bit 3 set? (SE)

```
541 \pIIE@qoval\z@{-\@ovyy}{-\@xdim}{-\@ovyy}\thr@\@tempdimc\@ovxx\z@
```

Now we've finished, draw the oval and finally close the group opened by \oval above.

```
542 \pIIE@strokeGraph
```

```
543 \endgroup}
```

\pIIE@qoval Ancillary macro; saves us some tokens above.

(PostScript: We could use the arc or arcto operator!)

```
544 \newcommand*\pIIE@qoval[8]{%
```

```
545 % \end{macrocode}
```

```
546 % Bit set?
```

```
547 % \begin{macrocode}
```

```
548 \ifodd\@tempcnta
```

```
549 \if@tempswa\pIIE@moveto{#1}{#2}\fi
```

```
550 \pIIE@lineto{#3}{#4}\pIIE@qcircle{#5}{#6}\pIIE@lineto{#7}{#8}%
```

```
551 \@tempswafalse
```

```
552 \else
```

```
553 \@tempswatrue
```

```
554 \fi
```

Shift by one bit.

```
555 \divide\@tempcnta\tw@}
```

\pIIE@get@quadrants According to the parameter (tlbr) bits are set in \@tempcnta:

0 = 1st Quadrant (NE) 1 = 2nd Quadrant (NW)

2 = 3rd Quadrant (SW) 3 = 4th Quadrant (SE)

(Cf. \@oval and \@ovvert in the L<sup>A</sup>T<sub>E</sub>X kernel.) We abuse \@setfpsbit from the float processing modules of the kernel.

```
556 \newcommand*\pIIE@get@quadrants[1]{%
```

```
557 \@ovttrue \@ovbtrue \@ovltrue \@ovrtrue \@tempcnta\z@
```

```
558 \@tfor\reserved@a:=#1\do{\csname @ov\reserved@a false\endcsname}%
```

```
559 \if@ovr \if@ovb\@setfpsbit2\fi \if@ovt\@setfpsbit4\fi \fi
```

```
560 \if@ovl \if@ovb\@setfpsbit1\fi \if@ovt\@setfpsbit8\fi \fi}
```

### 3.10.5 Quadratic Bezier Curve

\@bezier If #1=0 the primitive operators of the (back-end) format are used. The kernel version of \@bezier uses \put internally, which features \killglue and \ignorespaces commands in turn (at the beginning and end, respectively). Since we don't use \put, we have to add the latter commands by hand.

```
561 \def\@bezier#1(#2,#3)(#4,#5)(#6,#7){%
```

```
562 \ifnum #1=\z@
```

$$P_0 = (\#2, \#3) \quad P_m = (\#4, \#5) \quad P_3 = (\#6, \#7)$$

```
563 \killglue
```

```
564 \begingroup
```

```
565 \@defaultunitsset\@ovxx{#2}\unitlength
```

```
566 \@defaultunitsset\@ovyy{#3}\unitlength
```

```
567 \@defaultunitsset\@ovdx{#4}\unitlength
```

```
568 \@defaultunitsset\@ovdy{#5}\unitlength
```

```
569 \@defaultunitsset\@xdim{#6}\unitlength
```

```
570 \@defaultunitsset\@ydim{#7}\unitlength
```

$$P_1 = P_m + 1/3(P_0 - P_m)$$

```
571 \pIIE@bezier@QtoC\@ovxx\@ovdx\@ovro
```

```
572 \pIIE@bezier@QtoC\@ovyy\@ovdy\@ovri
```

$$P_2 = P_m + 1/3(P_3 - P_m)$$

```

573 \pIIE@bezier@QtoC\@xdim\@ovdx\@clnwd
574 \pIIE@bezier@QtoC\@ydim\@ovdy\@clnht
      (P0x, P0y)
575 \pIIE@moveto\@ovxx\@ovyy
      (P1x, P1y) (P2x, P2y) (P3x, P3y)
576 \pIIE@curveto\@ovro\@ovri\@clnwd\@clnht\@xdim\@ydim
577 \pIIE@strokeGraph
578 \endgroup
579 \ignorespaces
580 \else
581 \pIIE@old@bezier{#1}{#2,#3}{#4,#5}{#6,#7}
582 \fi}

```

`\pIIE@bezier@QtoC` Ancillary macro; saves us some tokens above.  
Transformation: quadratic bezier parameters  $\rightarrow$  cubic bezier parameters.  
(Missing: Reference for mathematical formula. Or is this trivial?)

```

583 \newcommand*\pIIE@bezier@QtoC[3]{%
584 \@tempdimc#1\relax \advance\@tempdimc-#2\relax
585 \divide\@tempdimc\thr@@ \advance\@tempdimc #2\relax
586 #3\@tempdimc}

```

### 3.10.6 Circle arcs

We need some auxiliary dimensions.

```

587 \ifx\undefined\@arclen \newdimen\@arclen \fi
588 \ifx\undefined\@arcrad \newdimen\@arcrad \fi
589 \ifx\undefined\@tempdimd \newdimen\@tempdimd \fi

```

`\pIIE@arc` #1: 0 (implicit) if we connect arc with a current point, 1 if we start drawing by this arc, 2 if we continue drawing. Other parameters: coordinates of the center (dimensions), radius (dimension), initial and final angle. If the final angle is greater then the initial angle, we “draw” in the positive sense (anticlockwise) otherwise in the negative sense (clockwise). First we check whether the radius is not negative and reduce the rotation to the interval  $[-720, 720]$ .

```

590 \newcommand*\pIIE@arc[6][0]{%
591 \@arcrad #4\relax
592 \ifdim \@arcrad<\z@ \pIIE@badcircarg \else
593 \@arclen #6\p@ \advance\@arclen -#5\p@
594 \ifdim \@arclen<\z@ \def\sign{-}\else\def\sign{}\fi
595 \ifdim \sign\@arclen>720\p@
596 \PackageWarning {pict2e}{The arc angle is reduced to -720..720}%
597 \@whiledim \sign\@arclen>720\p@ \do {\advance\@arclen-\sign360\p@}%
598 \@tempdima #5\p@ \advance\@tempdima \@arclen
599 \edef\@angleend{\strip@pt\@tempdima}%
600 \pIIE@@arc{#1}{#2}{#3}{#4}{#5}{\@angleend}%
601 \else
602 \pIIE@@arc{#1}{#2}{#3}{#4}{#5}{#6}%
603 \fi
604 \fi}

```

If the angle (its absolute value) is too large, the arc is recursively divided into 2 parts until the angle is at most 90 degrees.

```

605 \newcommand*\pIIE@@arc[6]{%
606 \begingroup
607 \ifdim \sign\@arclen>90\p@
608 \divide\@arclen 2
609 \@tempdima #5\p@ \advance\@tempdima \@arclen

```

```

610 \edef\@anglemid{\strip@pt\@tempdima}%
611 \def\@temp{\pIIE@arc{#1}{#2}{#3}{#4}{#5}}%
612 \expandafter\@temp\expandafter{\@anglemid}%
613 \def\@temp{\pIIE@arc{2}{#2}{#3}{#4}}%
614 \expandafter\@temp\expandafter{\@anglemid}{#6}%
615 \else

```

We approximate the arc by a Bezier curve. First we calculate the coordinates of the initial point:

```

616 \CalculateSin{#5}\CalculateCos{#5}%
617 \@tempdima\UseCos{#5}\@arcrad \advance\@tempdima #2\relax
618 \@tempdimb\UseSin{#5}\@arcrad \advance\@tempdimb #3\relax

```

The coordinates are added to the path if and how necessary:

```

619 \ifcase #1\relax
620 \pIIE@lineto\@tempdima\@tempdimb
621 \or \pIIE@moveto\@tempdima\@tempdimb
622 \or
623 \else \PackageWarning {pict2e}%
624 {Illegal obligatory argument in \protect\circlearc.}%
625 \fi

```

The distance of control points from the endpoints is  $\frac{4}{3}r \tan \frac{\varphi}{4}$  ( $\varphi$  is the angle and  $r$  is the radius of the arc).

```

626 \@tempdimc\@arclen \divide\@tempdimc\@iv
627 \edef\@angle{\strip@pt\@tempdimc}\CalculateTan{\@angle}%
628 \@linelen\UseTan{\@angle}\@arcrad \@linelen4\@linelen \divide\@linelen\thr@@

```

Coordinates of the first control point, added to the path:

```

629 \advance\@tempdima-\UseSin{#5}\@linelen
630 \advance\@tempdimb \UseCos{#5}\@linelen
631 \pIIE@add@nums\@tempdima\@tempdimb

```

Coordinates of the endpoint:

```

632 \CalculateSin{#6}\CalculateCos{#6}%
633 \@tempdima \UseCos{#6}\@arcrad \advance\@tempdima #2\relax
634 \@tempdimb \UseSin{#6}\@arcrad \advance\@tempdimb #3\relax

```

Coordinates of the second control point:

```

635 \@tempdimc \UseSin{#6}\@linelen \advance\@tempdimc \@tempdima
636 \@tempdimd-\UseCos{#6}\@linelen \advance\@tempdimd \@tempdimb

```

Adding the second control point and the endpoint to the path

```

637 \pIIE@add@nums\@tempdimc\@tempdimd
638 \pIIE@add@CP\@tempdima\@tempdimb
639 \pIIE@addtoGraph\pIIE@curveto@op
640 \fi
641 \endgroup}

```

`\arc` The `\arc` command generalizes (except that the radius instead of the diameter is used) the standard `\circle` adding as an obligatory first parameter comma separated pair of angles (initial and final). We start with `\pIIEarc` to avoid conflicts with other packages.

```

642 \newcommand*\pIIEarc
643 {\@ifstar{\@tempswatrue\pIIE@arc@}{\@tempswafalse\pIIE@arc@}}
644 \newcommand*\pIIE@arc@[2][0,360]{\pIIE@arc@@{#1}{#2}}
645 \def\pIIE@arc@@{#1,#2}#3{%
646 \if@tempswa
647 \pIIE@moveto\z@\z@
648 \@defaultunitsset\@tempdimc{#3}\unitlength
649 \pIIE@arc{\z@}{\z@}{\@tempdimc}{#1}{#2}%
650 \pIIE@closepath\pIIE@fillGraph

```



```

651 \else
652 \@defaultunitsset\@tempdimc{#3}\unitlength
653 \pIIE@arc[1]{\z@}{\z@}{\@tempdimc}{#1}{#2}%
654 \pIIE@strokeGraph
655 \fi}
656 \ifx\undefined\arc
657 \else
658 \PackageWarning{pict2e}{\protect\arc\space redefined}%
659 \fi
660 \let\arc\pIIEarc

```

### 3.10.7 Line, Vector, polyline, polyvector, and polygon

`\Line` We use recursive macros for `\polyline`, `\polyvector`, and `\polygon`.

```

\polyline 661 \let\lp@r(\let\rp@r)
\Vector 662 \def\Line(#1,#2)(#3,#4){\polyline(#1,#2)(#3,#4)}
\polyvector 663 \def\polyline(#1,#2){%
\polygon 664 \killglue
665 \@defaultunitsset\@tempdimc{#1}\unitlength
666 \@defaultunitsset\@tempdimb{#2}\unitlength
667 \pIIE@moveto{\@tempdimc}{\@tempdimb}%
668 \@ifnextchar\lp@r{\@polyline}{\PackageWarning{pict2e}%
669 {Polygonal lines require at least two vertices!}%
670 \ignorespaces}}
671 \def\@polyline(#1,#2){%
672 \@defaultunitsset\@tempdimc{#1}\unitlength
673 \@defaultunitsset\@tempdimb{#2}\unitlength
674 \pIIE@lineto{\@tempdimc}{\@tempdimb}%
675 \@ifnextchar\lp@r{\@polyline}{\pIIE@strokeGraph\ignorespaces}}
676 \def\Vector(#1,#2)(#3,#4){\polyvector(#1,#2)(#3,#4)}
677 \def\polyvector(#1,#2){%
678 \killglue
679 \@ifnextchar\lp@r{\begingroup\@polyvector(#1,#2)}{%
680 \PackageWarning{pict2e}%
681 {Polygonal vectors require at least two vertices!}\ignorespaces}}
682 \def\@polyvector(#1,#2)(#3,#4){%

```

See the similar definition for `\vector` (3.10.2)

```

683 \@defaultunitsset\@xdim{#1}\unitlength
684 \@defaultunitsset\@tempdima{#3}\unitlength
685 \@defaultunitsset\@ydim{#2}\unitlength
686 \@defaultunitsset\@tempdimb{#4}\unitlength
687 \advance\@tempdima-\@xdim \advance\@tempdimb-\@ydim
688 \ifdim\@tempdima=\z@ \@linelen\@tempdimb \else
689 \ifdim\@tempdimb=\z@ \@linelen\@tempdima \else
690 \pIIE@pyth\@tempdima\@tempdimb\@linelen
691 \fi
692 \fi
693 \ifdim\@linelen<\z@ \@linelen-\@linelen\fi
694 \pIIE@divide{\@tempdima}\@linelen\@ovxx
695 \pIIE@divide{\@tempdimb}\@linelen\@ovyy

```

Note the shift to the previous point in addition to the rotation.

```

696 \pIIE@concat\@ovxx\@ovyy{-\@ovyy}\@ovxx\@xdim\@ydim
697 \pIIE@vector \pIIE@fillGraph
698 \@ifnextchar\lp@r{\@polyvector(#3,#4)}{\endgroup\ignorespaces}}
699 \def\polygon{%
700 \killglue
701 \@ifstar{\begingroup\@tempwatrue\@polygon}%
702 {\begingroup\@tempwafalse\@polygon}}

```

```

703 \def\@polygon(#1,#2){%
704   \@defaultunitsset\@tempdimc{#1}\unitlength
705   \@defaultunitsset\@tempdimb{#2}\unitlength
706   \pIIE@moveto{\@tempdimc}{\@tempdimb}%
707   \@ifnextchar\lp@r{\@@polygon}{\PackageWarning{pict2e}%
708     {Polygons require at least two vertices!}}%
709   \ignorespaces}}
710 \def\@@polygon(#1,#2){%
711   \@defaultunitsset\@tempdimc{#1}\unitlength
712   \@defaultunitsset\@tempdimb{#2}\unitlength
713   \pIIE@lineto{\@tempdimc}{\@tempdimb}%
714   \@ifnextchar\lp@r{\@@polygon}{\pIIE@closepath
715     \if@tempswa\pIIE@fillGraph\else\pIIE@strokeGraph\fi
716     \endgroup
717     \ignorespaces}}

```

### 3.10.8 Path commands

\moveto Direct access to path constructions in PostScript and PDF.

```

\lineto 718 \def\moveto(#1,#2){%
\curveto 719   \@killglue
720   \@defaultunitsset\@tempdimc{#1}\unitlength
\circlearc 721   \@defaultunitsset\@tempdimb{#2}\unitlength
\closepath 722   \pIIE@moveto{\@tempdimc}{\@tempdimb}%
\strokepath 723   \ignorespaces}
\fillpath 724   \def\lineto(#1,#2){%
725     \@killglue
726     \@defaultunitsset\@tempdimc{#1}\unitlength
727     \@defaultunitsset\@tempdimb{#2}\unitlength
728     \pIIE@lineto{\@tempdimc}{\@tempdimb}%
729     \ignorespaces}
730   \def\curveto(#1,#2)(#3,#4)(#5,#6){%
731     \@killglue
732     \@defaultunitsset\@tempdimc{#1}\unitlength
733     \@defaultunitsset\@tempdimb{#2}\unitlength
734     \@defaultunitsset\@tempdima{#3}\unitlength
735     \@defaultunitsset\dimen@{#4}\unitlength
736     \@defaultunitsset\@ovxx{#5}\unitlength
737     \@defaultunitsset\@ovyy{#6}\unitlength
738     \pIIE@curveto{\@tempdimc}{\@tempdimb}{\@tempdima}{\dimen@}%
739     {\@ovxx}{\@ovyy}%
740     \ignorespaces}
741   \newcommand*\circlearc[6][0]{%
742     \@killglue
743     \@defaultunitsset\@tempdimc{#2}\unitlength
744     \@defaultunitsset\@tempdimb{#3}\unitlength
745     \@defaultunitsset\@tempdima{#4}\unitlength
746     \pIIE@arc[#1]{\@tempdimc}{\@tempdimb}{\@tempdima}{#5}{#6}%
747     \ignorespaces}
748   \def\closepath{\pIIE@closepath}
749   \def\strokepath{\pIIE@strokeGraph}
750   \def\fillpath{\pIIE@fillGraph}

```

### 3.10.9 Ends of paths, joins of subpaths

\buttcap Ends of paths and joins of subpaths in PostScript and PDF.

```

\roundcap 751 \ifcase\pIIE@mode\relax
\squarecap 752 \or
\miterjoin
\roundjoin
\beveljoin

```

```

753 \newcommand*\pIle@linecap@op{setlinecap}
754 \newcommand*\pIle@linejoin@op{setlinejoin}
755 \or
756 \newcommand*\pIle@linecap@op{J}
757 \newcommand*\pIle@linejoin@op{j}
758 \fi
759 \def\pIle@linecap{}
760 \def\pIle@linejoin{}
761 \def\buttcap{\edef\pIle@linecap{ 0 \pIle@linecap@op}}
762 \def\roundcap{\edef\pIle@linecap{ 1 \pIle@linecap@op}}
763 \def\squarecap{\edef\pIle@linecap{ 2 \pIle@linecap@op}}
764 \def\miterjoin{\edef\pIle@linejoin{ 0 \pIle@linejoin@op}}
765 \def\roundjoin{\edef\pIle@linejoin{ 1 \pIle@linejoin@op}}
766 \def\beveljoin{\edef\pIle@linejoin{ 2 \pIle@linejoin@op}}

```

## 3.11 Commands from other packages

### 3.11.1 Package ebezier

One feature from [3].

`\cbezier` #1, the maximum number of points to use, is simply ignored, as well as `\qbeziermax`.  
`\@cbezier` Like the kernel version of `\@bezier`, the original version of `\@cbezier` uses `\put` internally,  
`\pIle@cbezier` which features `\killglue` and `\ignorespaces` commands in turn (at the beginning and end, respectively). Since we don't use `\put`, we have to add the latter commands by hand.  
Original head of the macro:

```
\def\cbezier{\@ifnextchar [{\@cbezier}{\@cbezier[0]}}
```

Changed analogous to the L<sup>A</sup>T<sub>E</sub>X kernel's `\qbezier` and `\bezier`:

```

767 \AtBeginDocument{\@ifundefined{cbezier}{\newcommand}{\renewcommand}*\%
768 \cbezier[2][0]{\pIle@cbezier[#1]#2}%
769 \@ifdefinable\pIle@cbezier{}%
770 \def\pIle@cbezier#1#2(#3)#4(#5)#6({\@cbezier#1)(#3)(#5){}%
771 \def\@cbezier[#1](#2,#3)(#4,#5)(#6,#7)(#8,#9){%
772 \@killglue
773 \@defaultunitsset\@tempdimc{#2}\unitlength
774 \@defaultunitsset\@tempdimb{#3}\unitlength
775 \pIle@moveto{\@tempdimc}{\@tempdimb}%
776 \@defaultunitsset\@tempdimc{#4}\unitlength
777 \@defaultunitsset\@tempdimb{#5}\unitlength
778 \@defaultunitsset\@tempdima{#6}\unitlength
779 \@defaultunitsset\dimen@{#7}\unitlength
780 \@defaultunitsset\@ovxx{#8}\unitlength
781 \@defaultunitsset\@ovyy{#9}\unitlength
782 \pIle@curveto{\@tempdimc}{\@tempdimb}%
783 {\@tempdima}{\dimen@}{\@ovxx}{\@ovyy}%
784 \pIle@strokeGraph
785 \ignorespaces}%
786 }

```

### 3.11.2 Other packages

Other macros from various packages may be included in future versions of this package.

## 3.12 Mode ‘original’

Other branch of the big switch, started near the beginning of the code (see page 16).

```
787 \else
```

```

\oval   Gobble the new optional argument and continue with saved version. \maxovalrad is there to
\maxovalrad avoid error messages in case the user's document redefines it with \renewcommand*. Likewise,
\OriginalPictureCmds \OriginalPictureCmds is only needed for test documents.
788   \renewcommand*\oval[1] [] {\pIIe@oldoval}
789   \newcommand*\maxovalrad{20pt}
790   \newcommand*\OriginalPictureCmds{}
791 \fi

```

### 3.13 Final clean-up

Restore Catcodes.

```

792 \Gin@codes
793 \let\Gin@codes\relax
794 \</package>

```

## Acknowledgements

We would like to thank Michael Wichura for granting us permission to use his implementation of the algorithm for “pythagorean addition” from his `PiCTEX` package. Thanks go to Michael Vulis (MicroPress) for hints regarding a driver for the `VTeX` system. Walter Schmidt has reviewed the documentation and code, and has tested the `VTeX` driver. The members of the “`TeX-Stammtisch`” in Berlin, Germany, have been involved in the development of this package as our guinea pigs, i.e., alpha-testers; Jens-Uwe Morawski and Herbert Voss have also been helpful with many suggestions and discussions. Thanks to Claudio Beccari (`curve2e`) for some macros and testing. Thanks to Petr Olšák for some macros.

Finally we thank the members of The `LATEX` Team for taking the time to evaluate our new implementation of the picture mode commands, and eventually accepting it as the “official” `pict2e` package, as well as providing the `README` file.

## References

- [1] Leslie Lamport: *L<sup>A</sup>T<sub>E</sub>X – A Document Preparation System*, 2nd ed., 1994
- [2] Michel Goossens, Frank Mittelbach, Alexander Samarin: *The L<sup>A</sup>T<sub>E</sub>X Companion*, 1993
- [3] Gerhard A. Bachmaier: *The ebezier package*. CTAN: `macros/latex/contrib/ebezier/`, 2002
- [4] Michael Wichura: *The PiCTEX package*. CTAN: `graphics/pictex`, 1987
- [5] David Carlisle: *The pspicture package*. CTAN: `macros/latex/contrib/carlisle/`, 1992
- [6] David Carlisle: *The trig package*. CTAN: `macros/latex/required/graphics/`, 1999
- [7] Kresten Krab Thorup: *The pspic package*. CTAN: `macros/latex209/contrib/misc/`, 1991
- [8] Timothy Van Zandt: *The pstricks bundle*. CTAN: `graphics/pstricks/`, 1993, 1994, 2000

## Change History

v0.1a	v0.1d
General: First version. (RN) . . . . . 1	\pIIe@drawGraph: “gsave/grestore”

added. (RN) . . . . .	17	<code>\pIe@def@UL</code> : Check for negative or zero radius argument (RN,HjG) . . . . .	29
v0.1g		v0.2j	
<code>\pIe@circle</code> : Changed code (using <code>\pIe@add@qcircle</code> ). (HjG,RN) . . .	27	General: First release to CTAN (2004/02/19 v0.2j). (LaTeX Team) . .	1
<code>\pIe@drawGraph</code> : “gsave/grestore” removed. (RN) . . . . .	17	v0.2k	
v0.1h		General: Better control for indexing temporary registers while debugging (HjG) . . . . .	1
<code>\pIe@addtoGraph</code> : Added newline code (to be improved eventually). (RN,HjG) . . . . .	17	Better control over funny pagestyle while debugging (HjG) . . . . .	1
v0.1i		v0.2l	
<code>\pIe@drawGraph</code> : “gsave/grestore” restored for PDF (see ‘p2e-drivers.dtx’). (RN) . . . . .	17	<code>\line</code> : Macro added (RN/HjG) . . . . .	23
v0.1t		General: Even better control over funny pagestyle while debugging (RN) . . . .	1
<code>\pIe@get@quadrants</code> : Rename <code>\pIe@get@ovalquadrants</code> to <code>\pIe@get@quadrants</code> (RN) . . . . .	30	v0.2n	
v0.1u		<code>\pIe@circ</code> : Allow zero diameter (RN/HjG) . . . . .	27
<code>\@bezier</code> : Change calculation of cubic bezier parameters to use less tokens (HjG) . . . . .	30	<code>\pIe@def@UL</code> : Moved radius test to <code>\oval</code> , where it belongs (RN/HjG) . .	29
<code>\pIe@qcircle</code> : New ancillary macro (HjG) . . . . .	28	<code>\pIe@oval</code> : Allow zero diameter (RN/HjG) . . . . .	29
<code>\pIe@add@CP</code> : Rename <code>\pIe@add@XY</code> to <code>\pIe@add@CP</code> (HjG) . . . . .	18	Moved radius test from <code>\pIe@def@UL</code> (RN/HjG) . . . . .	29
<code>\pIe@bezier@QtoC</code> : New ancillary macro (HjG) . . . . .	31	General: Second release to CTAN (2004/04/22 v0.2n). (RN/HjG) . . . .	1
<code>\pIe@drawGraph</code> : Clear current point after output (HjG) . . . . .	18	v0.2o	
<code>\pIe@qcircle</code> : Change coding of quadrant number to match bit number in <code>\pIe@get@quadrants</code> (HjG) . . . . .	27	<code>\@bezier</code> : Supply <code>\ignorespaces</code> to match kernel version (HjG) . . . . .	30
<code>\pIe@qoval</code> : New ancillary macro (HjG)	30	<code>\@cbezier</code> : Supply <code>\ignorespaces</code> to match kernel version (HjG) . . . . .	35
<code>\pIe@vector</code> : New ancillary macro (HjG) . . . . .	24	<code>\Gin@codes</code> : Save and restore catcodes (HjG) . . . . .	13
<code>\pIe@vector@ltx</code> : New ancillary macro (HjG) . . . . .	25	<code>\line</code> : Use <code>\pIe@checkslopeargs</code> (HjG)	23
<code>\pIe@vector@pst</code> : New ancillary macro (HjG) . . . . .	26	<code>\vector</code> : Use <code>\pIe@checkslopeargs</code> (HjG) . . . . .	24
v0.1v		General: Third release to CTAN (2004/06/25 v0.2o). (RN/HjG) . . . .	1
<code>\pIe@qcircle</code> : Exchange <code>\@xdim</code> and <code>\@ydim</code> to <code>\@ovri</code> and <code>\@ovro</code> (HjG)	27	v0.2p	
v0.1w		<code>\@bezier</code> : <code>\@killglue</code> added. (RN) . . .	30
<code>\pIe@qoval</code> : Rename <code>\pIe@oval</code> to <code>\pIe@qoval</code> (HjG) . . . . .	30	<code>\@cbezier</code> : <code>\@killglue</code> added. (RN) . .	35
General: Index use of temporary registers while debugging (HjG) . . . . .	1	General: Fourth release to CTAN (2004/07/28 v0.2p). (RN) . . . . .	1
v0.1x		v0.2q	
<code>\pIe@FAI</code> : Introduce “inset”. (RN,HjG)	14	General: Fourth release to CTAN (2004/08/06 v0.2q). (RN/HjG) . . . .	1
v0.1y		v0.2r	
<code>\pIe@vector@ltx</code> : First implementation. (RN,HjG) . . . . .	25	<code>\pIe@pyth</code> : Two wrong global assignments changed. (RN) . . . . .	21
v0.2h		v0.2t	
<code>\pIe@badcircarg</code> : New error message. (RN,HjG) . . . . .	28	<code>\line</code> : All lines by <code>\@sline</code> (JT) . . . .	23
<code>\pIe@circ</code> : Check for negative or zero diameter argument (RN,HjG) . . . . .	27	v0.2u	
		General: Fifth release to CTAN (2008/06/29 v0.2u). (JT) . . . . .	1
		v0.2v	
		General: Sixth release to CTAN (2008/07/19 v0.2v). (JT) . . . . .	1

v0.2w	(2016/02/05 v0.3b). (RN) . . . . .	1
General: Seventh release to CTAN	New option ‘luatex’ (RN) . . . . .	14
(2008/07/19 v0.2w). (JT) . . . . .	v0.3c	
v0.2x	<code>\pIIE@checkslopeargs</code> : <code>\edef</code> for	
General: Eighth release to CTAN	parameters stored in macros –	
(2009/08/08 v0.2x). (JT) . . . . .	suggested by Phelype Oleinik (RN) . . . . .	23
v0.2y	v0.3d	
General: Ninth release to CTAN	<code>\pIIE@oval</code> : Allow spaces after the first	
(2011/04/05 v0.2y). (JT) . . . . .	optional Argument suggested by FMI	
v0.2z	(RN) . . . . .	29
General: 10th release to CTAN	<code>\pIIE@sline</code> : Simulated bounding boxes	
(2011/04/05 v0.2z). (JT) . . . . .	for <code>\line</code> and <code>\vector</code> suggested by	
v0.3a	Donald Arseneau (RN) . . . . .	23
<code>\pIIE@circle</code> : Changed code, <code>closepath</code>	<code>\pstarrows</code> : New user-level macros	
seems to be necessary. . . . .	<code>\ltxarrows</code> and <code>\pstarrows</code> . (RN) . . . . .	15
General: 11th release to CTAN	v0.3e	
(2016/01/09 v0.3a). (JT) . . . . .	General: Added <code>\Vector</code> and	
v0.3b	<code>\polyvector</code> suggested by FMI. (RN) . . . . .	33
General: 12th release to CTAN		

## Index

Numbers written in *italic* refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in *roman* refer to the code lines where the entry is used.

<b>Symbols</b>	732, 733, 734, 735,	<code>\@width</code> . . . . .	399, 400
<code>\!</code> . . . . .	736, 737, 743, 744,	<code>\^</code> . . . . .	3, 8
<code>\"</code> . . . . .	745, 773, 774, 776,	<b>A</b>	
<code>\%</code> . . . . .	777, 778, 779, 780, 781	<code>\arc</code> . . . . .	9, 502, <u>642</u>
<code>\*</code> . . . . .	<code>\@depth</code> . . . . .	<code>\arc*</code> . . . . .	9
<code>\:</code> . . . . .	399, 400	<b>B</b>	
<code>\@@polygon</code> . . . . .	<code>\@dot</code> . . . . .	<code>\begin</code> . . . . .	547
<code>\@angle</code> . . . . .	105, 118, <u>472</u>	<code>\beveljoin</code> . . . . .	10, <u>751</u>
<code>\@angleend</code> . . . . .	<code>\@gobble</code> . . . . .	<code>\bezier</code> . . . . .	8
<code>\@anglemid</code> . . . . .	74	<code>\box</code> . . . . .	376, 422
<code>\@arclen</code> . . . . .	<code>\@height</code> . . . . .	<code>\buttcap</code> . . . . .	10, 51, 478, <u>751</u>
593, 594, 595, 597,	<code>\@ifnextchar</code> . . . . .	<b>C</b>	
598, 607, 608, 609, 626	519, 668,	<code>\CalculateCos</code> . . . . .	235, 616, 632
<code>\@arccrad</code> . . . . .	675, 679, 698, 707, 714	<code>\CalculateSin</code> . . . . .	234, 616, 632
617, 618, 628, 633, 634	<code>\@ifstar</code> . . . . .	<code>\CalculateTan</code> . . . . .	627
<code>\@badlinearg</code> . . . . .	643, 701	<code>\catcode</code> . . . . .	3, 4, 5, 6, 7, 8, 10
358, 363, 364, 371, 406	<code>\@killglue</code> . . . . .	<code>\cbezier</code> . . . . .	8, <u>767</u>
<code>\@bezier</code> . . . . .	. 563, 664, 678, 700,	<code>\circle</code> . . . . .	6, 502
<code>\@cbezier</code> . . . . .	719, 725, 731, 742, 772	<code>\circle*</code> . . . . .	6
<code>\@circle</code> . . . . .	<code>\@makeother</code> . . . . .	<code>\circlearc</code> . . . . .	9, 503, 624, <u>718</u>
104, 117, <u>471</u>	9, 11, 12	<code>\closepath</code> . . . . .	10, <u>718</u>
<code>\@defaultunits</code> . . . . .	<code>\@nnil</code> . . . . .	<code>\countdef</code> . . . . .	319, 320
14	14	<code>\curveto</code> . . . . .	9, <u>718</u>
. 13, 370, 405, 408,	<code>\@oval</code> . . . . .	<b>D</b>	
474, 507, 525, 526,	112, 122, 519, <u>524</u>	<code>\Den</code> . . . . .	319, 329,
565, 566, 567, 568,	<code>\@polygon</code> . . . . .	330, 337, 345, 347, 349	
569, 570, 648, 652,	701, 702, 703	<code>\Denom</code> 318, 321, 322, 324, 329	
665, 666, 672, 673,	<code>\@polyline</code> . . . . .		
683, 684, 685, 686,	668, 671, 675		
704, 705, 711, 712,	<code>\@polyvector</code> . . . . .		
720, 721, 726, 727,	679, 682, 698		
	<code>\@setfpsbit</code> . . . . .		
	559, 560		
	<code>\@sline</code> . . . . .		
	102, 114		
	<code>\@temp</code> . . . . .		
	611, 612, 613, 614		
	<code>\@tempa</code> . . . . .		
	356,		
	357, 360, 361, 362, 363		
	<code>\@tempboxa</code> . . . . .		
	376, 398, 422		
	<code>\@tempdimd</code> . . . . .		
	589, 636, 637		
	<code>\@undefined</code> . . . . .		
	39, 40		
	<code>\@whiledim</code> . . . . .		
	597		
	<code>\@whilenum</code> . . . . .		
	339, 345		

<code>\dimendef</code> . . . . .	318	<b>P</b>	<code>\pIIE@def@UL</code> . . . . .	506	
<code>\dimexpr</code> . . . . .	14	<code>\pdfliteral</code> . . . . .	40, 46	<code>\pIIE@defaultUL</code> . .	506, 521
<b>E</b>		<code>\pIIE@@arc</code> . . . . .	600, 602, 605, 611, 613	<code>\pIIE@divide</code> . . . . .	297, 313, 316, 390, 412, 417, 418, 455, 694, 695
<code>\empty</code> . . . . .	361	<code>\pIIE@@cbezier</code> . . . . .	767	<code>\pIIE@drawGraph</code> . . . . .	165, 166, 167
<code>\end</code> . . . . .	545	<code>\pIIE@@divide</code> . . . . .	339, 344	<code>\pIIE@FAI</code> . . . . .	54, 433, 454
<b>F</b>		<code>\pIIE@@pyth</code> . . . . .	305, 312	<code>\pIIE@FAL</code> . . . . .	54, 428, 450
<code>\fillpath</code> . . . . .	10, 718	<code>\pIIE@@qcircle</code> . . . . .	489, 491, 493, 495, 498	<code>\pIIE@FAW</code> . . . . .	54, 427, 449
<b>G</b>		<code>\pIIE@add@CP</code> . . . . .	182, 245, 247, 257, 271, 282, 638	<code>\pIIE@fill@op</code> . . . . .	140, 149, 171
<code>\Gin@codes</code> . . . . .	2, 792, 793	<code>\pIIE@add@num</code> . . . . .	195, 215	<code>\pIIE@fillGraph</code> . . . . .	165, 421, 478, 650, 697, 715, 750
<code>\Gin@driver</code> . . . . .	15, 21, 23, 24, 25, 26, 27, 28, 29, 32, 33, 34, 35, 83, 89, 90, 91, 92, 94	<code>\pIIE@add@nums</code> . . . . .	189, 207, 209, 211, 214, 216, 222, 224, 226, 253, 255, 265, 268, 278, 280, 631, 637	<code>\pIIE@get@quadrants</code> . . . . .	531, 556
<b>H</b>		<code>\pIIE@addtoGraph</code> . . . . .	159, 187, 193, 198, 205, 212, 214, 215, 216, 227, 245, 247, 258, 272, 283, 285, 639	<code>\pIIE@GRAPH</code> . . . . .	159, 179, 180
<code>\hbox</code> . . . . .	398	<code>\pIIE@arc</code> . . . . .	590, 649, 653, 746	<code>\pIIE@linecap</code> . . . . .	175, 759, 761, 762, 763
<b>I</b>		<code>\pIIE@arc@</code> . . . . .	643, 644	<code>\pIIE@linecap@op</code> . . . . .	753, 756, 761, 762, 763
<code>\I</code> . . . . .	320, 338, 339, 350	<code>\pIIE@arc@@</code> . . . . .	644, 645	<code>\pIIE@linejoin</code> . . . . .	175, 760, 764, 765, 766
<code>\ifpIIE@pdfliteral@ok</code> . . . . .	37, 50	<code>\pIIE@badcircarg</code> . . . . .	475, 500, 522, 592	<code>\pIIE@linejoin@op</code> . . . . .	754, 757, 764, 765, 766
<code>\ifx</code> . . . . .	39, 40, 361, 362, 587, 588, 589, 656	<code>\pIIE@bezier@Qtoc</code> . . . . .	434, 435, 436, 437, 571, 572, 573, 574, 583	<code>\pIIE@lineto</code> . . . . .	246, 374, 441, 442, 443, 444, 446, 460, 462, 463, 464, 465, 467, 469, 470, 550, 620, 674, 713, 728
<code>\ignorespaces</code> . . . . .	579, 670, 675, 681, 698, 709, 717, 723, 729, 740, 747, 785	<code>\pIIE@buttcap</code> . . . . .	49, 532	<code>\pIIE@lineto@op</code> . . . . .	140, 149, 247
<b>L</b>		<code>\pIIE@CAW</code> . . . . .	54, 427, 449	<code>\pIIE@maxovalrad</code> . . . . .	517, 528
<code>\Line</code> . . . . .	9, 661	<code>\pIIE@checkslopearg</code> . . . . .	356, 357, 359	<code>\pIIE@mode</code> . . . . .	15, 36, 80, 98, 99, 126, 131, 138, 139, 201, 248, 751
<code>\line</code> . . . . .	4, 101, 115, 365, 366	<code>\pIIE@checkslopeargs</code> . . . . .	351	<code>\pIIE@moveto</code> . . . . .	244, 373, 438, 459, 499, 549, 575, 621, 647, 667, 706, 722, 775
<code>\lineto</code> . . . . .	9, 718	<code>\pIIE@checkslopeargsline</code> . . . . .	351, 368	<code>\pIIE@moveto@op</code> . . . . .	140, 149, 245
<code>\lp@r</code> . . . . .	661, 668, 675, 679, 698, 707, 714	<code>\pIIE@checkslopeargsvector</code> . . . . .	353, 403	<code>\pIIE@old@bezier</code> . . . . .	101, 119, 581
<code>\ltxarrows</code> . . . . .	58, 64	<code>\pIIE@circ</code> . . . . .	471, 472, 473	<code>\pIIE@old@cbezier</code> . . . . .	101, 120
<b>M</b>		<code>\pIIE@circle</code> . . . . .	477, 480	<code>\pIIE@old@circle</code> . . . . .	101, 117
<code>\m@ne</code> . . . . .	339	<code>\pIIE@closepath</code> . . . . .	285, 482, 650, 714, 748	<code>\pIIE@old@dot</code> . . . . .	101, 118
<code>\maxdimen</code> . . . . .	325, 327, 332, 334	<code>\pIIE@closepath@op</code> . . . . .	140, 149, 285	<code>\pIIE@old@oval</code> . . . . .	101, 122
<code>\maxovalrad</code> . . . . .	8, 505, 520, 788	<code>\pIIE@code</code> . . . . .	15, 73, 74, 178	<code>\pIIE@old@sline</code> . . . . .	101, 114
<code>\miterjoin</code> . . . . .	10, 751	<code>\pIIE@concat</code> . . . . .	202, 218, 419, 696	<code>\pIIE@old@line</code> . . . . .	101, 115
<code>\moveto</code> . . . . .	9, 718	<code>\pIIE@concat@op</code> . . . . .	140, 149, 212, 227, 242	<code>\pIIE@old@oval</code> . . . . .	101, 121, 788
<b>N</b>		<code>\pIIE@CPx</code> . . . . .	180, 182, 263, 266, 269	<code>\pIIE@old@vector</code> . . . . .	101, 116
<code>\newdimen</code> . . . . .	587, 588, 589	<code>\pIIE@CPy</code> . . . . .	180, 182, 264, 267, 270	<code>\pIIE@oval</code> . . . . .	517
<code>\newif</code> . . . . .	37	<code>\pIIE@curveto</code> . . . . .	439, 447, 576, 738, 782	<code>\pIIE@pdfliteral</code> . . . . .	37
<code>\Num</code> . . . . .	319, 329, 330, 337, 345, 346, 347, 349, 350	<code>\pIIE@curveto@op</code> . . . . .	140, 149, 272, 283, 639	<code>\pIIE@pdfliteral@okfalse</code> . . . . .	41
<code>\Numb</code> . . . . .	320, 330, 331, 336, 337, 347, 348, 349			<code>\pIIE@pdfliteral@oktrue</code> . . . . .	38
<code>\number</code> . . . . .	336, 348			<code>\pIIE@PTtoBP</code> . . . . .	200, 217, 242
<code>\Numer</code> . . . . .	318, 321, 322, 323, 329			<code>\pIIE@pyth</code> . . . . .	286, 409, 690
<b>O</b>				<code>\pIIE@qcircle</code> . . . . .	481, 482, 483, 550
<code>\OriginalPictureCmds</code> . . . . .	113, 788			<code>\pIIE@qoval</code> . . . . .	535, 538, 539, 540, 541, 544
<code>\oval</code> . . . . .	7, 111, 121, 502, 517, 788				

<code>\pIle@rcurveto</code>	<a href="#">249</a> , <a href="#">260</a> , <a href="#">499</a>	<code>\pIle@vector</code>	.....	<b>S</b>	
<code>\pIle@rotate</code>	..... <a href="#">202</a> , <a href="#">218</a>	..	<a href="#">59</a> , <a href="#">62</a> , <a href="#">420</a> , <a href="#">425</a> , <a href="#">697</a>	<code>\setbox</code>	..... <a href="#">398</a>
<code>\pIle@scale</code>	..... <a href="#">202</a> , <a href="#">218</a>	<code>\pIle@vector@ltx</code>	.. <a href="#">59</a> , <a href="#">426</a>	<code>\sign</code>	..... <a href="#">323</a> , <a href="#">327</a> , <a href="#">334</a> ,
<code>\pIle@scale@PTtoBP</code>	.....	<code>\pIle@vector@pst</code>	.. <a href="#">62</a> , <a href="#">448</a>		<a href="#">342</a> , <a href="#">594</a> , <a href="#">595</a> , <a href="#">597</a> , <a href="#">607</a>
.....	<a href="#">169</a> , <a href="#">202</a> , <a href="#">218</a>	<code>\pIlearc</code>	..... <a href="#">642</a> , <a href="#">660</a>	<code>\squarecap</code>	..... <a href="#">10</a> , <a href="#">751</a>
<code>\pIle@setlinewidth@op</code>	.	<code>\polygon</code>	..... <a href="#">9</a> , <a href="#">661</a>	<code>\strokepath</code>	..... <a href="#">10</a> , <a href="#">718</a>
.....	<a href="#">140</a> , <a href="#">149</a> , <a href="#">174</a>	<code>\polygon*</code>	..... <a href="#">9</a>	<b>T</b>	
<code>\pIle@sline</code>	... <a href="#">372</a> , <a href="#">379</a> , <a href="#">407</a>	<code>\polyline</code>	..... <a href="#">9</a> , <a href="#">661</a>	<code>\tempend</code>	..... <a href="#">342</a> , <a href="#">343</a>
<code>\pIle@stroke@op</code>	<a href="#">140</a> , <a href="#">149</a> , <a href="#">176</a>	<code>\polyvector</code>	..... <a href="#">9</a> , <a href="#">661</a>	<b>U</b>	
<code>\pIle@strokeGraph</code>	. <a href="#">166</a> ,	<code>\pstarrows</code>	..... <a href="#">58</a> , <a href="#">65</a>	<code>\undefined</code>	<a href="#">587</a> , <a href="#">588</a> , <a href="#">589</a> , <a href="#">656</a>
	<a href="#">375</a> , <a href="#">478</a> , <a href="#">542</a> , <a href="#">577</a> ,	<b>Q</b>		<code>\UseCos</code>	<a href="#">237</a> , <a href="#">617</a> , <a href="#">630</a> , <a href="#">633</a> , <a href="#">636</a>
	<a href="#">654</a> , <a href="#">675</a> , <a href="#">715</a> , <a href="#">749</a> , <a href="#">784</a>	<code>\Q</code>	... <a href="#">325</a> , <a href="#">332</a> , <a href="#">336</a> , <a href="#">342</a> , <a href="#">348</a>	<code>\UseSin</code>	<a href="#">236</a> , <a href="#">618</a> , <a href="#">629</a> , <a href="#">634</a> , <a href="#">635</a>
<code>\pIle@tempa</code>	..... <a href="#">18</a> ,	<code>\qbezier</code>	..... <a href="#">8</a>	<code>\UseTan</code>	..... <a href="#">628</a>
	<a href="#">233</a> , <a href="#">234</a> , <a href="#">235</a> , <a href="#">236</a> ,	<code>\qbeziermax</code>	..... <a href="#">8</a>	<b>V</b>	
	<a href="#">237</a> , <a href="#">300</a> , <a href="#">301</a> , <a href="#">306</a> , <a href="#">307</a>	<b>R</b>		<code>\Vector</code>	..... <a href="#">9</a> , <a href="#">661</a>
<code>\pIle@tempb</code>	<a href="#">18</a> , <a href="#">236</a> , <a href="#">238</a> , <a href="#">239</a>	<code>\roundcap</code>	..... <a href="#">10</a> , <a href="#">751</a>	<code>\vector</code>	<a href="#">5</a> , <a href="#">103</a> , <a href="#">116</a> , <a href="#">365</a> , <a href="#">401</a>
<code>\pIle@tempc</code>	<a href="#">18</a> , <a href="#">237</a> , <a href="#">238</a> , <a href="#">239</a>	<code>\roundjoin</code>	..... <a href="#">10</a> , <a href="#">751</a>	<code>\vrule</code>	..... <a href="#">399</a> , <a href="#">400</a>
<code>\pIle@translate</code>	.. <a href="#">202</a> , <a href="#">218</a>	<code>\rp@r</code>	..... <a href="#">661</a>		