

CreEPS — Creating EPS
Version
2.00

Uwe Fabricius & Thomas Pohl

December 6, 2009

Contents

1	Introduction	3
2	Units	3
3	Namespace and Parameter Types	3
4	Constructing the canvas	3
5	Drawing	4
5.1	Drawing in non-path mode	5
5.2	Drawing in path mode	6
6	The Attributes	11
6.1	Introduction	11
6.2	Attribute Reference	12
6.2.1	Line Attributes	12
6.2.2	Colors	16
6.2.3	Fonts	16
6.2.4	Filling Patterns	18
7	Transformations	19
8	LaTeX font output	21
9	Embedding external EPS files	23
10	Pitfalls	23
11	Contact and Download	24
12	Thanks	25
13	Acknowledgment	25
14	License (MIT license)	25
15	In a Nutshell	26

1 Introduction

`CreEPS` is a C++ class which provides an easy-to-use interface for generating vector drawings and stores them as encapsulated PostScript¹ (EPS) files.

2 Units

`CreEPS` expects all length units to be in millimeters (mm) except for the font sizes which have to be specified in points as e. g. in `LATEX`. Angles have to be specified in degree, can be negative and may also exceed 360 degrees. Color or gray scale values have to lie between 0 and 1.

3 Namespace and Parameter Types

The `CreEPS` classes are embedded in namespace `ns_creeps`.

The generic data types of parameters passed to `CreEPS` can be configured in header file `CreEPS_Types.hpp`. The default definition of types is:

```
typedef int      CreEPS_INT;
typedef bool     CreEPS_BOOL;
typedef char     CreEPS_CHAR;
typedef float    CreEPS_FLOAT;
typedef double   CreEPS_DOUBLE;
```

This configuration, however, does not influence the data types used by `CreEPS` internally, but are simply intended to provide an interface corresponding to the data types used outside of `CreEPS`.

4 Constructing the canvas

As always in C++ you have to construct a `CreEPS` object before you can do anything. You have the choice between three different constructors:

1. `CreEPS::CreEPS(const CreEPS_CHAR* filename,`
 `CreEPS_FLOAT x1, CreEPS_FLOAT y1,`
 `CreEPS_FLOAT x2, CreEPS_FLOAT y2`

¹The name PostScript[®] is a registered trademark of Adobe Systems Inc.

- ```
[, CreEPS_BOOL latexOutput
 [, const char* altEPSFilename]])
```
2. `CreEPS::CreEPS( const char* filename,
 CreEPS_FLOAT width, CreEPS_FLOAT height
 [, CreEPS_BOOL latexOutput
 [, const char* altEPSFilename] ] )`
  3. `CreEPS::CreEPS()`

The first two flavors expect a file name as the first argument. If the file already exists it will be overwritten without any warnings. The first constructor expects the coordinates of the lower left corner and the upper right corner. In the second constructor you just have to specify the width and the height of the bounding box. The lower left corner will automatically be located at (0,0).

Using the optional boolean `latexOutput` you can switch from the default PS font output (`false`) to  $\text{\LaTeX}$  font output (`true`). Please refer to section 8 for more details about  $\text{\LaTeX}$  font output.

If you use the third constructor which does not expect any arguments you have to call `CreEPS::newFile` which accepts exactly the same arguments as the first two constructors before you can use `CreEPS` to draw anything.

In the case that you have finished one EPS file and want to start drawing the next one just call `CreEPS::newFile`. The old file will be finalized and the new one is ready for drawing.

Although you should never have to call the destructor `~CreEPS` explicitly, there is no problem if you do so. The destructor just adds some lines to the file informing the PS interpreter that this is the end of the file and closes it afterwards. `CreEPS::finalize` does the same as the destructor, but you can still create a new file with the same `CreEPS` object by calling `CreEPS::newFile` afterwards.

## 5 Drawing

You have to choose between two exclusive modes when drawing on the canvas, but you can switch from one to the other as you wish.

## 5.1 Drawing in non-path mode

In this mode every call of a drawing method is an atomic action, i. e. two consecutive calls do not interfere in any way. Following is an overview of all available drawing methods in this mode with a diagram showing its usage ([Cat a] stands for an optional attribute argument that will be described in section 6):

- `line` ( CreEPS\_FLOAT x1, CreEPS\_FLOAT y1,  
CreEPS\_FLOAT x2, CreEPS\_FLOAT y2 [, CAT a] )
- `rectStroke` ( CreEPS\_FLOAT x , CreEPS\_FLOAT y ,  
CreEPS\_FLOAT width, CreEPS\_FLOAT height [, CAT a] )
- `rectFill` ( CreEPS\_FLOAT x , CreEPS\_FLOAT y ,  
CreEPS\_FLOAT width, CreEPS\_FLOAT height [, CAT a] )
- `curve` ( CreEPS\_FLOAT x0, CreEPS\_FLOAT y0,  
CreEPS\_FLOAT x1, CreEPS\_FLOAT y1,  
CreEPS\_FLOAT x2, CreEPS\_FLOAT y2,  
CreEPS\_FLOAT x3, CreEPS\_FLOAT y3 [, CAT a] )
- `arc` ( CreEPS\_FLOAT x , CreEPS\_FLOAT y ,  
CreEPS\_FLOAT radius, CreEPS\_FLOAT alpha,  
CreEPS\_FLOAT beta [, CAT a] )
- `circle` ( CreEPS\_FLOAT x , CreEPS\_FLOAT y ,  
CreEPS\_FLOAT radius [, CAT a] )
- `ellipse` ( CreEPS\_FLOAT x , CreEPS\_FLOAT y ,  
CreEPS\_FLOAT radius1, CreEPS\_FLOAT radius2  
[, CreEPS\_FLOAT rot] [, CAT a] )
- `ellipseArc` ( CreEPS\_FLOAT x , CreEPS\_FLOAT y ,  
CreEPS\_FLOAT radius1, CreEPS\_FLOAT radius2,  
CreEPS\_FLOAT alpha, CreEPS\_FLOAT beta  
[, CreEPS\_FLOAT rot] [, CAT a] )
- `disk` ( CreEPS\_FLOAT x , CreEPS\_FLOAT y ,  
CreEPS\_FLOAT radius [, CAT a] )
- `print` ( CreEPS\_FLOAT x , CreEPS\_FLOAT y ,  
[float alpha,] const char\* text [, CAT a] )

- `printf` ( [Cat a,] CreEPS\_FLOAT x , CreEPS\_FLOAT y , [float alpha,], const char\* format, ... )

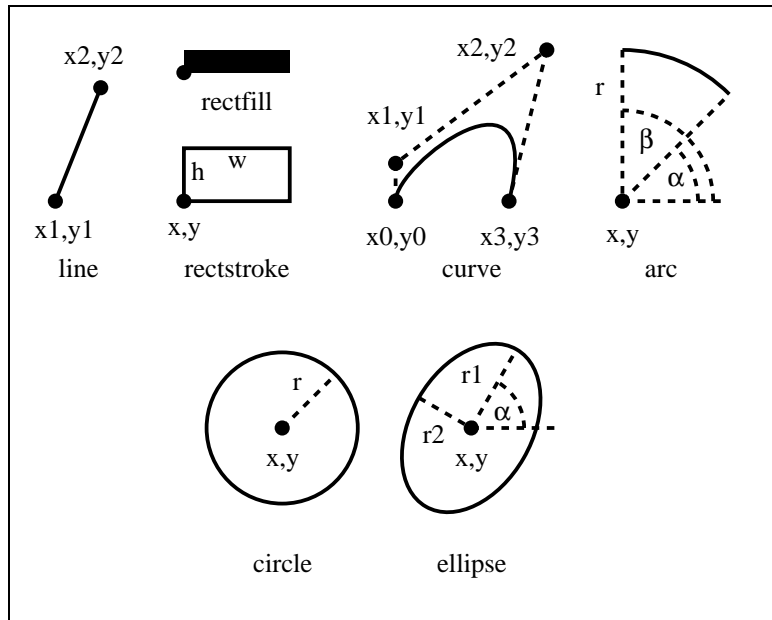


Figure 1: Examples for non-path drawing methods

The method `disk` is just a filled `circle`. The usage of the other drawing methods should be clear from their arguments and the illustration.

**Note:** Since the overall number of arguments in the `printf` method is not fixed, the optional attribute argument has a special location compared to the other methods. Both printing methods `print` and `printf` accept an optional argument specifying the text angel after the coordinate arguments.

## 5.2 Drawing in path mode

All the aforementioned non-path drawing methods instantly draw something on the canvas. In path mode however, you first create a path by calling methods that are similar to the non-path ones. A path can consist of several subpaths, i. e. paths that are not connected. After constructing the complete path which by itself does not alter the canvas, you can use it in different ways: You can stroke the path and/or fill its shape as often as you want. If you have no more use for this particular path, you leave the path drawing

mode which automatically deletes the path. Then you can start constructing a new path or use the non-path drawing methods.

You switch to the path drawing mode and start the construction of a new path by calling one of these two methods:

- `startPath()`
- `startPath( CreEPS_FLOAT x, CreEPS_FLOAT y )`

The second method lets you specify the starting point of the new path. It is equivalent to calling the first method and the `addMove` method afterwards. Following is a list of all available path mode methods that add an element to the current path:

- `addMove` ( CreEPS\_FLOAT x, CreEPS\_FLOAT y )
- `addRelativeMove( CreEPS_FLOAT dx, CreEPS_FLOAT dy )`
- `addLine` ( CreEPS\_FLOAT x, CreEPS\_FLOAT y )
- `addRelativeLine( CreEPS_FLOAT dx, CreEPS_FLOAT dy )`
- `addArc` ( CreEPS\_FLOAT x, CreEPS\_FLOAT y,  
CreEPS\_FLOAT radius,  
CreEPS\_FLOAT alpha, CreEPS\_FLOAT beta )  
Adds an arc in counter-clockwise direction to the path.
- `addArcN` ( CreEPS\_FLOAT x, CreEPS\_FLOAT y,  
CreEPS\_FLOAT radius,  
CreEPS\_FLOAT alpha, CreEPS\_FLOAT beta )  
Does the same as the method above but in clockwise direction.
- `addCircle` ( CreEPS\_FLOAT x, CreEPS\_FLOAT y,  
CreEPS\_FLOAT radius )
- `addEllipse` ( CreEPS\_FLOAT x, CreEPS\_FLOAT y,  
CreEPS\_FLOAT radius1, CreEPS\_FLOAT radius2  
[, CreEPS\_FLOAT rot] )
- `addEllipseArc` ( CreEPS\_FLOAT x, CreEPS\_FLOAT y,  
CreEPS\_FLOAT radius1, CreEPS\_FLOAT radius2,  
CreEPS\_FLOAT alpha, CreEPS\_FLOAT beta  
[, CreEPS\_FLOAT rot] )

- `addEllipseArcN` ( `CreEPS_FLOAT x`, `CreEPS_FLOAT y`,  
`CreEPS_FLOAT radius1`, `CreEPS_FLOAT radius2`,  
`CreEPS_FLOAT alpha`, `CreEPS_FLOAT beta`  
`[, CreEPS_FLOAT rot]` )
- `addArcT` ( `CreEPS_FLOAT x1`, `CreEPS_FLOAT y1`,  
`CreEPS_FLOAT x2`, `CreEPS_FLOAT y2`,  
`CreEPS_FLOAT radius` )

Appends an arc of a circle to the current path, possibly preceded by a straight line segment. The arc is defined by the `radius` and two tangent lines. The tangent lines are those drawn from the current point  $(x_0, y_0)$  to  $(x_1, y_1)$ , and from  $(x_1, y_1)$  to  $(x_2, y_2)$  (see Fig. 2).<sup>2</sup>

- `addArcTLine` ( `CreEPS_FLOAT x1`, `CreEPS_FLOAT y1`,  
`CreEPS_FLOAT x2`, `CreEPS_FLOAT y2`,  
`CreEPS_FLOAT radius` )

Does the same as `addArcT` plus drawing a line from the end of the arc to  $(x_2, y_2)$ .

- `addCurve` ( `CreEPS_FLOAT x1`, `CreEPS_FLOAT y1`,  
`CreEPS_FLOAT x2`, `CreEPS_FLOAT y2`,  
`CreEPS_FLOAT x3`, `CreEPS_FLOAT y3` )

- `closeSubpath()`

Adds a straight line to the current path from the current point to the starting point of the current subpath. See Fig. 3 for illustration.

In the PS language there is the notion of the *current point*, which is the point where the last path construction method stopped on the canvas. The methods `addRelativeMove`, `addLine`, `addRelativeLine`, `addArcT`, `addArcTLine`, and `addCurve` rely on the existence of such a current point. If it has not been set by a preceding path construction method, the PS interpreter (e. g. GhostScript or your printer) will complain and stop displaying the EPS file. It is therefore necessary to set a current point using one of the other path mode drawing methods or the `startPath` method with specified coordinates.

Except for the methods `addCircle` and `addEllipse`, all path mode drawing methods connect the current point with the path they describe by a straight line. If there is no current point set, there won't be a connection line.

---

<sup>2</sup>Taken from the *PostScript Language Reference Manual*, 2. Edition, Addison-Wesley

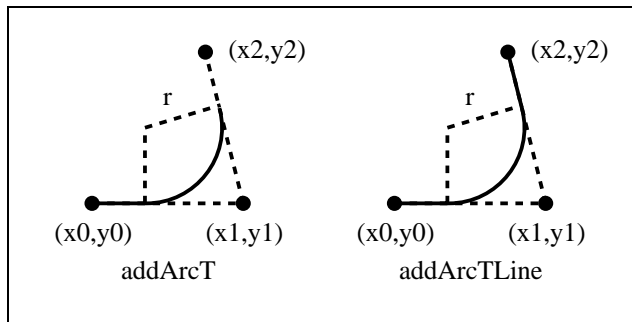


Figure 2: An example for the path drawing methods `addArcT` and `addArcTLine`.

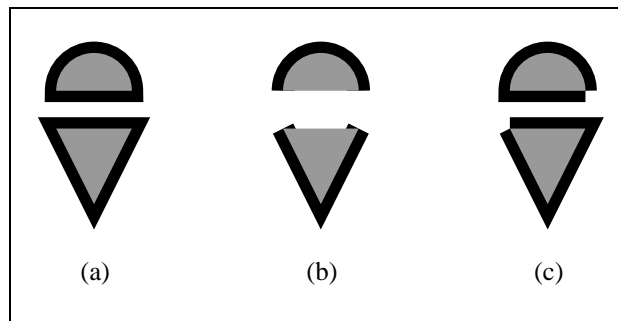


Figure 3: Difference between closed and non-closed strokes in the path mode: closed stroke using `closeSubpath` (a), non-closed stroke (b), manually closed stroke using `addLine` (c). Manually closing a path can create a strange artifact at the endpoints, so use `closeSubpath` instead.

So far, you just know how to construct a path which does not draw anything on the canvas. In order to use the path and to draw something you can call the method `usePath` at any point between the calling of `startPath` and `endPath`:

- `usePath( DrawMode mode [, CA t a] )`  
 Uses the path that you have constructed so far to draw something on the canvas depending on the chosen draw mode. `CreEPS::STROKE` draws a stroke along the path, `CreEPS::FILL` fills its shape completely, and `CreEPS::EOFILL` fills it according to some rule that is hard to explain with words (see Fig. 4). `CreEPS::CLIP` and `CreEPS::EOCLIP` set the current path as the clipping region for all further actions. Remove the current clipping region by calling `resetClipping()`.

- `endPath()`  
Just ends the current path.
- `endPath( DrawMode mode [, CAAt a] )`  
Does the same as a `usePath` call and ends the current path afterwards.
- `endPath( DrawMode mode1, DrawMode mode2  
[, CAAt a1 [, CAAt a2] ] )`  
Does exactly the same as the above `endPath` call, except for the fact that you can specify two different actions. The two optional attribute arguments refer to the first respectively the second action. The order of the two actions does make a difference (see Fig. 4).

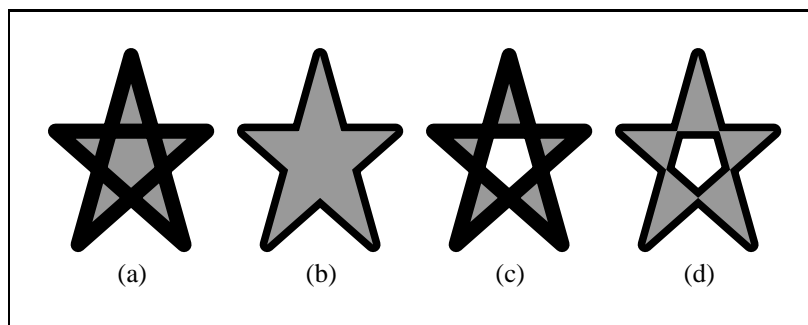


Figure 4: In all four examples the path was constructed in the same fashion. In (a) the path was first filled with `CreEPS::FILL` and then drawn, in (b) it was done vice versa. For (c) and (d) the filling command was replaced with `CreEPS::EOFILL`. In this drawing mode the part of the shape that would be filled two, four, six, ... times is not filled at all.

In Fig. 5 the `usePath` method is called several times for the same path with different attributes. Have a look at the example source code for more details.

**Attention:** You must not use non-path drawing methods while being in path mode (i. e. between a `startPath` and its consecutive `endPath` call) and vice versa! `CreEPS` will print an error message and exit immediately if you do so.

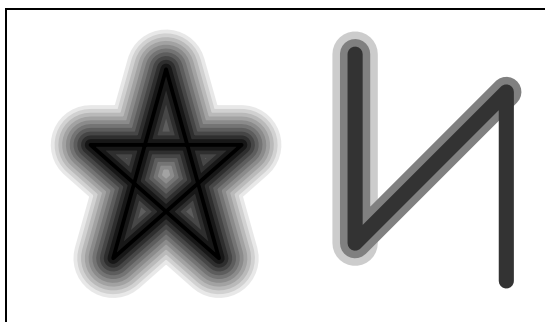


Figure 5: Some nice tricks with `usePath`. If you use `ghostview` under Linux you might have to turn off anti-aliasing to see a nicely shaded star.

## 6 The Attributes

### 6.1 Introduction

There are various attributes a single element of a drawing created with `CreEPS` can have. For example for a simple line we could specify a line thickness and a gray scale value. This is the job of the class `CAt` and its derived classes. All the derived classes have self-explanatory names and only a constructor and a destructor. You simply call the constructor of one of them to create a certain attribute. For example

```
CAtLineThickness(4);
```

creates a drawing attribute with the information "line thickness = 4mm", or

```
CAtGrayScale(0.5);
```

creates an attribute with a gray value of 0.5. But of course a line could have both attributes, a line thickness and a gray value. How to combine them? The properties of more than one `CAt` object can be merged into one object just by "catenating" them with the logical OR-operator "|". For example

```
CAtLineThickness(4) | CAtGrayScale(0.5);
```

creates an attribute object with line thickness equal to 4 and a gray scale value of 0.5. And now we just have to pass this object to one of the drawing routines, because they all have an optional parameter of the type `CAt&` (this

is the optional parameter [, `CAt a`] that has already been introduced and used in the previous sections. Assume that we already have created an instance of `CreEPS` named `creeps`. Then the following command will draw a 4mm thick line with gray value 0.5 from point (10,10) to (10,90):

```
creeps.line(10, 10, 10, 90, CAtLineThickness(4) |
 CAtGrayScale(0.5));
```

The properties of an attribute object passed to a drawing routine have a local scope and are valid only for this function. That means, if we draw a second line after the line above without specifying attributes, the global settings fix the appearance of the line. You can change these global settings with `CreEPS::setAttributes( CAt& )`. The following code creates the graphic below.

```
CreEPS creeps("cre.eps", 50, 30);
creeps.line(10, 5, 10, 25);
creeps.setAttributes(CAtGrayScale(0.6) | CAtLineThickness(3));
creeps.line(20, 5, 20, 25,
 CAtGrayScale(0.9) | CAtLineThickness(6));
creeps.line(30, 5, 30, 25);
creeps.line(40, 5, 40, 25, CAtGrayScale(0.3));
```



## 6.2 Attribute Reference

Here you can find a reference of the classes derived from `CAt`.

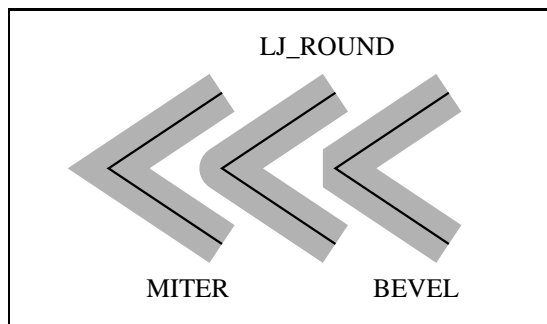
### 6.2.1 Line Attributes

```
CAtLineThickness(const CreEPS_FLOAT thickness)
```

Simply the thickness of all lines drawn as straight lines and curves. `thickness` specifies the thickness in millimeters.

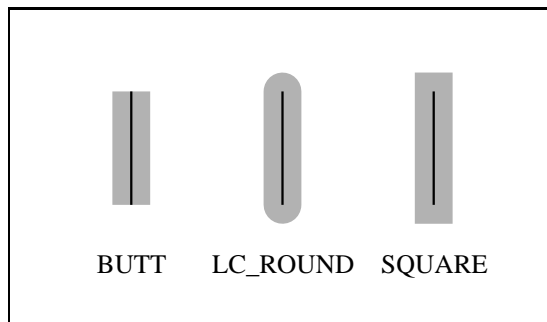
```
CAtLineJoin(const LINEJOIN linejoin)
```

The line join defines the way vertices of line paths are treated. `linejoin` can be one of the enumerations constants defined in `CAt`: `MITER`, `LJ_ROUND`, `BEVEL`. For their differences just have a look at the following example.



```
CAtLineCap(const LINECAP linecap)
```

Here `linecap` is one of the constants `CAt::BUTT`, `::LC_ROUND`, `::SQUARE` and sets the appearance of the line ends.



```
CAtLineDash(const CreEPS_CHAR *const dashstring,
 const CreEPS_FLOAT offset)
```

First, how could we *define* how a line is dashed? We could define for the whole length of a black line e.g. "draw 2mm black, than leave 2mm empty, than draw 1mm black, than ...". But in a sensibly dashed line this pattern should be repeated after a while. So for example we could define the dashed line by "2mm black, than 2mm space, than 1mm black, 2mm space, and now from the beginning". Fine, if we agree now upon always starting with "black" and also upon using mm as units, it would be enough to say "2 2 1

2". And exactly this is the string you have to pass as parameter `dashstring`. So the code line

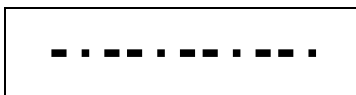
```
creeps.line(5, 5, 40, 5, CAtLineDash("2 2 1 2", 0));
```

draws this line (The second parameter `offset`, here 0, will be explained below. Please ignore it for the moment.)



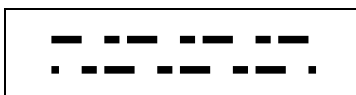
But what happens, if the string contains an odd number of numbers? Then we could imagine the creation of the line more like this: We copy the instruction string and glue the original and the copy together and then use the result as described above.

```
creeps.line(5, 5, 40, 5, CAtLineDash("2 2 1", 0));
```



The second parameter `offset` is just an offset you start from evaluating the dash rules. The rule "4 3 2 1" has the length  $4 + 3 + 2 + 1 = 11$ . And in this total length you start with the specified offset, i.e. you cut off the first `offset` millimeters.

```
creeps.line(5, 7, 40, 7, CAtLineDash("4 3 2 1", 0));
creeps.line(5, 3, 40, 3, CAtLineDash("4 3 2 1", 3));
```



In most cases there is no need for difficult line patterns. Therefore `CreEPS` offers four typical base patterns defined as constants in `CAt`: `SOLID`, `DOT`, `DASH`, `DOTDASH`, that you can pass to the second constructor of the class `CAtLineDash`:

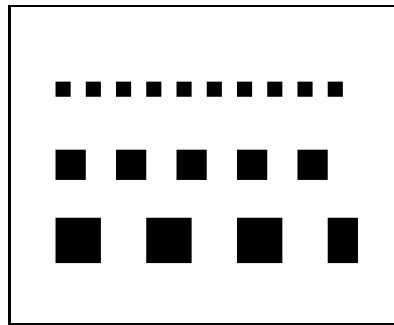
```
CAtLineDash(const LINEDASH dash,
 const CreEPS_FLOAT factor)
```

`CreEPS` creates the line pattern according to the line thickness so that for example the dots in pattern `DOT` are dots as you would expect.

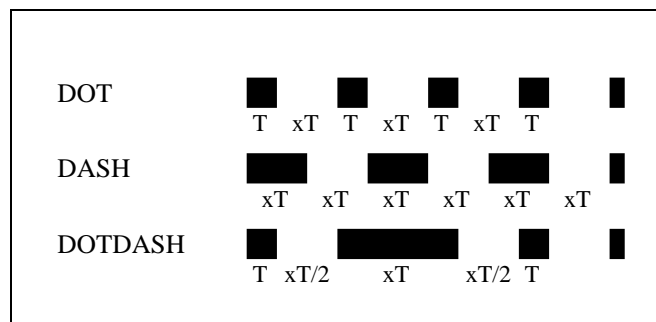
```

for(int i = 1; i < 4; i++) {
 creeps.line(5, 40-10*i, 45, 40-10*i,
 CAtLineDash(CAt::DOT, 0) | CAtLineThickness(2*i));
}

```



With the parameter **factor** ( $\geq 1$ ) the predefined pattern can be scaled in the following way. Let  $T$  denote the line thickness and  $x$  the parameter factor.



The patterns DOT, DASH, DOTDASH are also adapted to the line cap. Note that a globally set line dash (DOT, DASH, DOTDASH) is not updated, if the line thickness or line cap are changed. This might lead to strange effects, even that a line is not drawn at all (see example below). So setting the line dash globally must be used carefully, if the line thickness or the line cap change after that.

```

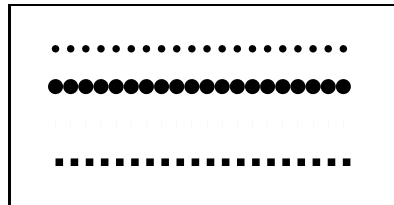
creeps.setAttributes(CAtLineThickness(1) |
 CAtLineCap(CAt::LC_ROUND) |
 CAtLineDash(CAt::DOT, 1));
creeps.line(5, 20, 45, 20);
creeps.line(5, 15, 45, 15, CAtLineThickness(2));

```

```

creeps.setAttributes(CAtLineCap(CAt::BUTT));
creeps.line(5, 10, 45, 10); // THIS LINE STAYS INVISIBLE !!
creeps.line(5, 5, 45, 5, CAtLineDash(CAt::DOT, 1));

```



### 6.2.2 Colors

```

CAtColor(const CreEPS_FLOAT red,
 const CreEPS_FLOAT green,
 const CreEPS_FLOAT blue)

```

Defines the RGB color curves, lines, fonts, ... are drawn with. `red`, `green`, `blue`  $\in [0,1]$ .

```

CAtGrayScale(const CreEPS_FLOAT grayscale)

```

Identical to `Color( grayscale, grayscale, grayscale )`.

```

CAtBackgroundColor(const CreEPS_FLOAT red,
 const CreEPS_FLOAT green,
 const CreEPS_FLOAT blue)

```

Defines the background color used in filling patterns ( $\rightarrow$  filling patterns).

```

CAtTransparentBackground()

```

The transparent background color in filling patterns. As the default background is transparent, this attribute is only needed, if a non-transparent background has been set before by `CreEPS::setAttributes`.

### 6.2.3 Fonts

```

CAtFont(const CreEPS_CHAR *fontstring [, const CreEPS_FLOAT scale]
)

```

```

CAtFont(const CreEPS_FLOAT scale)

```

In order to select a font you have to pass a string specifying a valid font. In addition to this you can scale the font to `scale` point size. So there are three different ways to use and create `Font`:

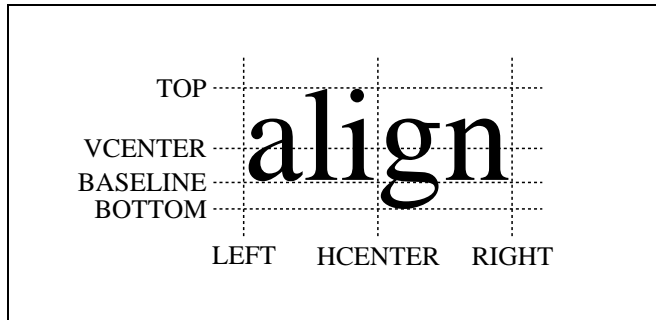
```
CAatFont("Times-Roman", 5)
 selects the font "Times-Roman" with size 5 pts
CAatFont("Times-Roman")
 selects "Times-Roman" and uses the current global font size
CAatFont(3)
 uses the current global font and scales it to 3 pts
```

Some possible font types are (here in size 10 pts):

|                                |                                     |                                   |
|--------------------------------|-------------------------------------|-----------------------------------|
| Times-Roman                    | Helvetica                           | Courier                           |
| <i>Times-Italic</i>            | <i>Helvetica-Oblique</i>            | <i>Courier-Oblique</i>            |
| <b>Times-Bold</b>              | <b>Helvetica-Bold</b>               | <b>Courier-Bold</b>               |
| <i><b>Times-BoldItalic</b></i> | <i><b>Helvetica-BoldOblique</b></i> | <i><b>Courier-BoldOblique</b></i> |

```
CAatTextAlignment(const CreEPS_INT alignment)
```

As a printed word has a finite width and height, it does not make sense to say "draw the word 'CreEPS' at point (10,10)". Should the midpoint of the "C" be placed at (10,10) or the midpoint of a box bounding the whole word? That's why you can specify text alignment in `CreEPS`. In `CAT` there are constants defined for the vertical (`TOP`, `VCENTER`, `BASELINE`, `BOTTOM`) and for the horizontal (`LEFT`, `HCENTER`, `RIGHT`) alignment. To define the point of a text which should lie on the coordinates specified in `CreEPS::printf` one horizontal alignment constant can be combined with one vertical constant with the binary OR operator `"|"`. For the meaning of the constants see the graphic below.



In addition to this there are the combined constants `CENTER = (HCENTER | VCENTER)` and `DEFAULT = (LEFT | BASELINE)` (initial value).

### 6.2.4 Filling Patterns

There are some predefined filling patterns paths can be filled with.

```
CAtHexDotFilling(const CreEPS_FLOAT radius,
 const CreEPS_FLOAT distance)
```

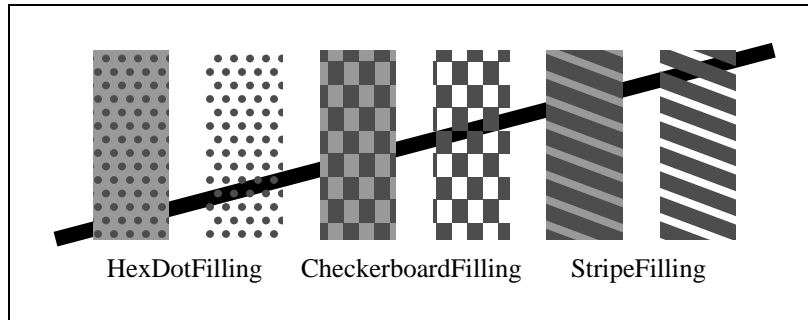
In a hexagonal grid dots with radius `radius` are placed. The mesh size of the grid, i.e. the distance between the midpoints of neighboring dots, is `distance`.

```
CAtCheckerboardFilling(const CreEPS_FLOAT x,
 const CreEPS_FLOAT y)
```

Just a checkerboard with field width `x` and field height `y`.

```
CAtStripeFilling(const CreEPS_FLOAT width,
 const CreEPS_FLOAT distance,
 const CreEPS_INT angle [, CAt a])
```

Stripes with width `width` and distance `distance` between each other and the slope angle `angle`.



The filling pattern is drawn with the color specified with `CAtColor` on a background specified with `CAtBackgroundColor`. The default background is transparent. If you have set a global background, this can be changed only by using or setting `CAtTransparentBackground()`.

## 7 Transformations

In the PostScript language all graphical output is transformed in a certain way before being drawn to the canvas. This makes it easy to scale or move things around on the canvas without altering the actual coordinates of the graphical elements. In a mathematical notation the given coordinates are transformed by the so called current transformation matrix (CTM):

$$\text{CTM} = \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

$$x \mapsto ax + by + t_x$$

$$y \mapsto cx + dy + t_y$$

Do not let the mathematical notation scare you off the really useful features that transformations offer. The following five easy-to-use methods described below do what you expect them to do even without understanding the equations.

- `CreEPS::applyRotation( CreEPS_FLOAT alpha )`  
Rotates the following output by the given angle. The pivoting point is the current origin of the coordinate system:

$$\text{CTM} \mapsto \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \text{CTM}$$

- `CreEPS::applyTranslation( CreEPS_FLOAT dx, CreEPS_FLOAT dy )`  
Shifts the origin of the coordinate system by the given values:

$$\text{CTM} \mapsto \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{bmatrix} \times \text{CTM}$$

- `CreEPS::applyScaling( CreEPS_FLOAT sx [, CreEPS_FLOAT sy] )`  
Scales the following output with the given values in x and y direction separately. If you specify just one parameter the scaling will be isotropic:

$$\text{CTM} \mapsto \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \text{CTM}$$

- `CreEPS::applyTransformation( CreEPS_FLOAT m[3][2] )`  
A generic transformation matrix can be specified which will be multiplied with the CTM:

$$\text{CTM} \mapsto \begin{bmatrix} m[0][0] & m[0][1] & 0 \\ m[1][0] & m[1][1] & 0 \\ m[2][0] & m[2][1] & 1 \end{bmatrix} \times \text{CTM}$$

There is one mathematical restriction for the given transformation matrix: It has to be invertible. Since this is the case for all practical transformations you should not worry too much about this.

- `CreEPS::resetTransformations()`  
Resets the CTM to the matrix that was set right at the beginning of the current EPS file. This is not equal to setting the unity matrix, since a surrounding PS file might change the CTM to place the EPS file at a certain position of the page. This method takes care of that issue and make sure that everything is placed where it should be.
- `CreEPS::saveTransformation()`  
Stores the current CTM for later use. For example you could save the CTM, do another transformation and after some drawing load the old CTM again by calling ...
- `CreEPS::loadTransformation()`  
But be careful: Do not try to load more transformations than you have saved before. `CreEPS` does some bookkeeping and informs you if you do

so. It might also be a good idea to make sure that you load all transformations again before closing the file. Violating this rule will most likely do no harm, but who knows. Calling `CreEPS::resetTransformations()` does not remove or alter the stored transformations.

If you use more than one transformation, e. g. a translation and a rotation, the order does matter. Mathematically speaking, the multiplication of the transformation matrices is not commutative.

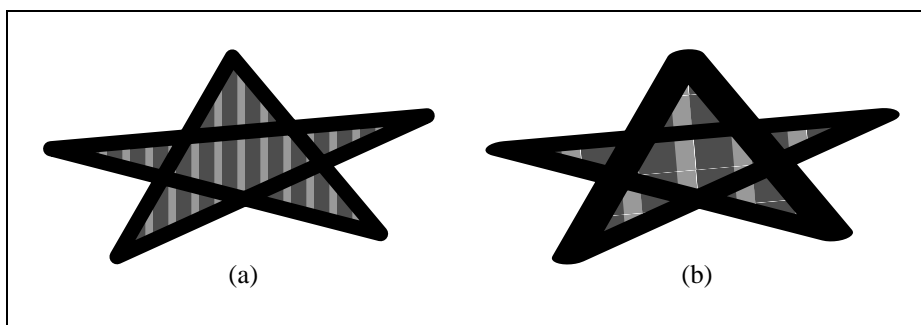


Figure 6: The path for both examples was constructed while a scaling transformation was active. For (a) the transformation was reset before stroking and filling the path, for (b) it remained active.

Transformations do not only alter the coordinates of graphical elements, but also affect line strokes and fillings as you can see in Fig. 6. Depending on the effect that you want to achieve it might therefore be necessary to turn off transformations between constructing the path and using it.

## 8 $\text{\LaTeX}$ font output

When you create a new `CreEPS` object you can select  $\text{\LaTeX}$  output as mentioned in section 4.

- `CreEPS::CreEPS( const char* filename,  
CreEPS_FLOAT x1, CreEPS_FLOAT y1,  
CreEPS_FLOAT x2, CreEPS_FLOAT y2  
[, CreEPS_BOOL latexOutput  
[, const char* altEPSfilename] ] )`
- `CreEPS::CreEPS( const char* filename,  
CreEPS_FLOAT width, CreEPS_FLOAT height`

```
[, CreEPS_BOOL latexOutput
 [, const char* altEPSFilename]])
```

You just have to specify a `true` boolean for `latexOutput`. `CreEPS` then creates another file for you with the same filename as specified for `filename` appended with “\_t”. This file will contain all the text that you printed as a  $\LaTeX$  file<sup>3</sup>. Instead of using `\includegraphics{example.eps}` from the `graphics` package you have to include the generated  $\LaTeX$  file by using `\input{example.eps_t}`. However, it is mandatory to load two packages with the command `\usepackage{graphics,color}` in the including  $\LaTeX$  file.

The name of the EPS file that will be loaded from within the generated  $\LaTeX$  file is the given `filename` string by default. If you specify the optional `altEPSFilename` string, this will be used instead.

Let’s think of a scenario where this feature might come in handy: Your code that uses `CreEPS` lives inside the directory `~/doc/bin`, but you want `CreEPS` to store its generated files in `~/doc`. To do so you create a `CreEPS` object with `CreEPS creeps("../foo.eps", 20, 10, true);`.

`CreEPS` will then generate two files in `~/doc`, namely `foo.eps` and `foo.eps_t`. In your  $\LaTeX$  source you just have to write `\input{foo.eps_t}`. The problem is that this  $\LaTeX$  file will then try to load the EPS file `../foo.eps` since this is what you specified in the constructor. To avoid this you can give an optional alternative name for the EPS file that should be loaded from within the generated  $\LaTeX$ file. In our case the constructor call

```
CreEPS creeps("../foo.eps", 200, 100, true, "foo.eps");
```

would do the trick.

If you choose to use  $\LaTeX$  font output the font selection via `CAtFont` will only change the size of the  $\LaTeX$  font but not the font style itself (Times–Roman–Bold, Helvetica–Oblique, ...). Of course you can use the  $\LaTeX$  way of selecting the font within the printed string by the appropriate  $\LaTeX$  commands<sup>4</sup>. Rotating and aligning the text works exactly the same as in PS font output mode.

Another thing that you have to consider when using  $\LaTeX$  font output are transformations: Neither translation, scaling, rotation, nor a generic transformation will influence the position, the size or the orientation of the  $\LaTeX$  text.

---

<sup>3</sup>Of course you can have a look at the EPS file directly, but all the text will be missing.

<sup>4</sup>Do not forget to use two backslashes for  $\LaTeX$  commands in your C++ strings.

## 9 Embedding external EPS files

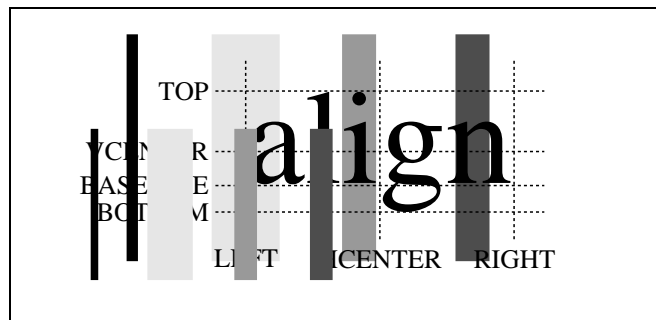
CreEPS provides the integration of external EPS files into the CreEPS file. To embed an external file call

```
CreEPS::embedEPS(const CreEPS_CHAR *const filename)
```

with `filename` being the name of the file you want to integrate. Attributes that have been set before in the CreEPS-generated file will not influence the drawing behavior of the imported file, except all modifications of the transformation matrix. So you might use `applyTranslation`, `applyRotation`, and `applyScaling` to position the embedded image.

Example:

```
CreEPS creeps("cre.eps", 85, 40);
creeps.applyScaling(3.0/2.0, 3.0/2.0);
creeps.embedEPS("lines.eps");
creeps.applyScaling(2.0/3.0, 2.0/3.0);
creeps.embedEPS("align.eps");
creeps.embedEPS("lines.eps");
```



## 10 Pitfalls

- Depending on the GhostScript version you use the colors of filled backgrounds may not be the expected.
- When you use filling patterns globally GhostScript uses that filling for line strokes as well.

Below you can find the details for a PS interpreter that we used and that showed the test picture above with the intended colors and filling:



Figure 7: These are some examples for the above-mentioned pitfalls. On the left-hand side you should see a blue box with red dots and white text. On the right-hand side you see the underlined word "Striped?". Depending on your PostScript interpreter you either see a striped text, a striped underline, or both (filled with yellow lines on magenta background). The text may appear scrambled until you zoom in or print the document.

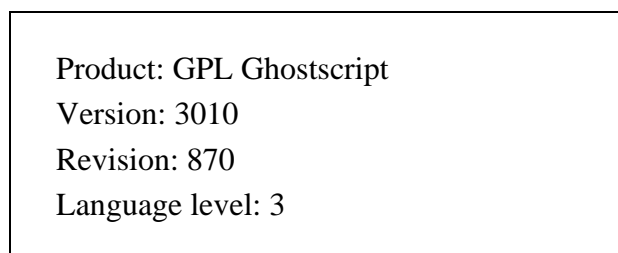


Figure 8: In this box you can read some information about the interpreter which you are using right now to display this manual.

|                 |                 |
|-----------------|-----------------|
| Product:        | GNU Ghostscript |
| Version:        | 3010            |
| Revision:       | 651             |
| Language level: | 3               |

## 11 Contact and Download

You are looking for the latest version of **CreEPS** or want to contact us? Then just visit our web pages:

- <http://uwefabricius.de/>
- <http://thomas-pohl.info/>

## 12 Thanks

Thanks to Marcus Mohr for proof reading this documentation and for being the first real user of CreEPS. This implies that he is the one to blame if there are any mistakes left ;-)

## 13 Acknowledgment

Part of the programming for CreEPS was done while working for the Simulation Group (<http://www10.informatik.uni-erlangen.de/>) at the University of Erlangen–Nuremberg (<http://www.uni-erlangen.de/>), Germany.

## 14 License (MIT license)

Copyright (c) 2002 - 2009 Uwe Fabricius, Thomas Pohl

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## 15 In a Nutshell

| CreEPS Constructors                                                                                                                                                 |  |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| CreEPS( char* filename, CreEPS_FLOAT x1, CreEPS_FLOAT y1, CreEPS_FLOAT x2, CreEPS_FLOAT y2<br>[, CreEPS_BOOL latexOutput [, char* altEPSfilename] ] )               |  |
| CreEPS( char* filename, CreEPS_FLOAT width, CreEPS_FLOAT height<br>[, CreEPS_BOOL latexOutput [, char* altEPSfilename] ] )                                          |  |
| CreEPS() [the only allowed call in this state is newFile]                                                                                                           |  |
| newFile( ... ) [accepts the same arguments as one of the first two constructors]                                                                                    |  |
| CreEPS Non-Path Drawing Methods                                                                                                                                     |  |
| line ( CreEPS_FLOAT x1, CreEPS_FLOAT y1, CreEPS_FLOAT x2, CreEPS_FLOAT y2 [, CAT a] )                                                                               |  |
| rectStroke( CreEPS_FLOAT x , CreEPS_FLOAT y , CreEPS_FLOAT width, CreEPS_FLOAT height [, CAT a] )                                                                   |  |
| rectFill ( CreEPS_FLOAT x , CreEPS_FLOAT y , CreEPS_FLOAT width, CreEPS_FLOAT height [, CAT a] )                                                                    |  |
| curve ( CreEPS_FLOAT x0, CreEPS_FLOAT y0, CreEPS_FLOAT x1, CreEPS_FLOAT y1,<br>CreEPS_FLOAT x2, CreEPS_FLOAT y2, CreEPS_FLOAT x3, CreEPS_FLOAT y3 [, CAT a] )       |  |
| arc ( CreEPS_FLOAT x , CreEPS_FLOAT y , CreEPS_FLOAT radius,<br>CreEPS_FLOAT alpha, CreEPS_FLOAT beta [, CAT a] )                                                   |  |
| circle ( CreEPS_FLOAT x , CreEPS_FLOAT y , CreEPS_FLOAT radius [, CAT a] )                                                                                          |  |
| ellipse ( CreEPS_FLOAT x , CreEPS_FLOAT y , CreEPS_FLOAT radius1, CreEPS_FLOAT radius2<br>[, CreEPS_FLOAT rot] [, CAT a] )                                          |  |
| ellipseArc( CreEPS_FLOAT x , CreEPS_FLOAT y , CreEPS_FLOAT radius1, CreEPS_FLOAT radius2,<br>CreEPS_FLOAT alpha, CreEPS_FLOAT beta [, CreEPS_FLOAT rot] [, CAT a] ) |  |
| disk ( CreEPS_FLOAT x , CreEPS_FLOAT y , CreEPS_FLOAT radius [, CAT a] )                                                                                            |  |
| print ( CreEPS_FLOAT x , CreEPS_FLOAT y , [float alpha,] char* text [, CAT a] )                                                                                     |  |
| printf ( [CAT a,] CreEPS_FLOAT x , CreEPS_FLOAT y , [float alpha,] char* format, ... )                                                                              |  |
| CreEPS Path Drawing Methods                                                                                                                                         |  |
| startPath()                                                                                                                                                         |  |
| startPath( CreEPS_FLOAT x, CreEPS_FLOAT y )                                                                                                                         |  |
| addMove ( CreEPS_FLOAT x, CreEPS_FLOAT y )                                                                                                                          |  |
| addRelativeMove( CreEPS_FLOAT dx, CreEPS_FLOAT dy )                                                                                                                 |  |
| addLine ( CreEPS_FLOAT x, CreEPS_FLOAT y )                                                                                                                          |  |
| addRelativeLine( CreEPS_FLOAT dx, CreEPS_FLOAT dy )                                                                                                                 |  |
| addArc ( CreEPS_FLOAT x, CreEPS_FLOAT y, CreEPS_FLOAT radius,<br>CreEPS_FLOAT alpha, CreEPS_FLOAT beta )                                                            |  |
| addArcN ( CreEPS_FLOAT x, CreEPS_FLOAT y, CreEPS_FLOAT radius,<br>CreEPS_FLOAT alpha, CreEPS_FLOAT beta )                                                           |  |
| addCircle ( CreEPS_FLOAT x, CreEPS_FLOAT y, CreEPS_FLOAT radius )                                                                                                   |  |
| addEllipse ( CreEPS_FLOAT x, CreEPS_FLOAT y, CreEPS_FLOAT radius1, CreEPS_FLOAT radius2<br>[, CreEPS_FLOAT rot] )                                                   |  |
| addEllipseArc ( CreEPS_FLOAT x, CreEPS_FLOAT y, CreEPS_FLOAT radius1, CreEPS_FLOAT radius2,<br>CreEPS_FLOAT alpha, CreEPS_FLOAT beta [, CreEPS_FLOAT rot] )         |  |
| addEllipseArcN ( CreEPS_FLOAT x, CreEPS_FLOAT y, CreEPS_FLOAT radius1, CreEPS_FLOAT radius2,<br>CreEPS_FLOAT alpha, CreEPS_FLOAT beta [, CreEPS_FLOAT rot] )        |  |
| addArcT ( CreEPS_FLOAT x1, CreEPS_FLOAT y1,<br>CreEPS_FLOAT x2, CreEPS_FLOAT y2, CreEPS_FLOAT radius )                                                              |  |
| addArcTLine ( CreEPS_FLOAT x1, CreEPS_FLOAT y1,<br>CreEPS_FLOAT x2, CreEPS_FLOAT y2, CreEPS_FLOAT radius )                                                          |  |
| addCurve ( CreEPS_FLOAT x1, CreEPS_FLOAT y1,<br>CreEPS_FLOAT x2, CreEPS_FLOAT y2, CreEPS_FLOAT x3, CreEPS_FLOAT y3 )                                                |  |
| closeSubpath ( )                                                                                                                                                    |  |
| usePath( DrawMode mode [, CAT a] )                                                                                                                                  |  |
| endPath()                                                                                                                                                           |  |
| endPath( DrawMode mode [, CAT a] )                                                                                                                                  |  |
| endPath( DrawMode mode1, DrawMode mode2 [, CAT a1 [, CAT a2] ] )                                                                                                    |  |
| CreEPS Transformations                                                                                                                                              |  |
| applyRotation ( CreEPS_FLOAT alpha )                                                                                                                                |  |
| applyTranslation ( CreEPS_FLOAT dx, CreEPS_FLOAT dy )                                                                                                               |  |
| applyScaling ( CreEPS_FLOAT sx [, CreEPS_FLOAT sy] )                                                                                                                |  |
| applyTransformation ( CreEPS_FLOAT m[3][2] )                                                                                                                        |  |
| resetTransformations()                                                                                                                                              |  |
| saveTransformation()                                                                                                                                                |  |
| loadTransformation()                                                                                                                                                |  |

| CreEPS Miscellaneous functions                                                   |  |
|----------------------------------------------------------------------------------|--|
| resetClipping()                                                                  |  |
| embedEPS( char* filename )                                                       |  |
| special( char* text, ... )                                                       |  |
| CreEPS Attributes: Cat                                                           |  |
| CatLineThickness( CreEPS_FLOAT thickness )                                       |  |
| CatLineJoin ( LINEJOIN linejoin )                                                |  |
| CatLineCap ( LINECAP linecap )                                                   |  |
| CatLineDash ( char* dashstring, CreEPS_FLOAT offset )                            |  |
| CatLineDash ( LINEDASH dash, CreEPS_FLOAT factor )                               |  |
| CatColor ( CreEPS_FLOAT red, CreEPS_FLOAT green, CreEPS_FLOAT blue )             |  |
| CatGrayScale ( CreEPS_FLOAT grayscale )                                          |  |
| CatFont ( CreEPS_CHAR *fontstring [, CreEPS_FLOAT scale] )                       |  |
| CatFont ( CreEPS_FLOAT scale )                                                   |  |
| CatTextAlignment( CreEPS_INT alignment )                                         |  |
| CatHexDotFilling ( CreEPS_FLOAT radius, CreEPS_FLOAT distance )                  |  |
| CatCheckerboardFilling ( CreEPS_FLOAT x, CreEPS_FLOAT y )                        |  |
| CatStripeFilling ( CreEPS_FLOAT width, CreEPS_FLOAT distance, CreEPS_INT angle ) |  |
| CatBackgroundColor ( CreEPS_FLOAT red, CreEPS_FLOAT green, CreEPS_FLOAT blue )   |  |
| CatTransparentBackground()                                                       |  |

| CreEPS Constants                  |                |               |                |              |                |
|-----------------------------------|----------------|---------------|----------------|--------------|----------------|
| DrawMode                          | CreEPS::STROKE | CreEPS::FILL  | CreEPS::EOFILL | CreEPS::CLIP | CreEPS::EOCLIP |
| Cat Constants                     |                |               |                |              |                |
| LINEJOIN                          | Cat::MITER     | Cat::LJ_ROUND | Cat::BEVEL     |              |                |
| LINECAP                           | Cat::BUTT      | Cat::LC_ROUND | Cat::SQUARE    |              |                |
| LINEDASH                          | Cat::SOLID     | Cat::DOT      | Cat::DASH      | Cat::DOTDASH |                |
| Text alignment                    | Cat::TOP       | Cat::VCENTER  | Cat::BASELINE  | Cat::BOTTOM  |                |
| (can be combined with   operator) | Cat::LEFT      | Cat::HCENTER  | Cat::RIGHT     |              |                |
|                                   | Cat::CENTER    | Cat::DEFAULT  |                |              |                |