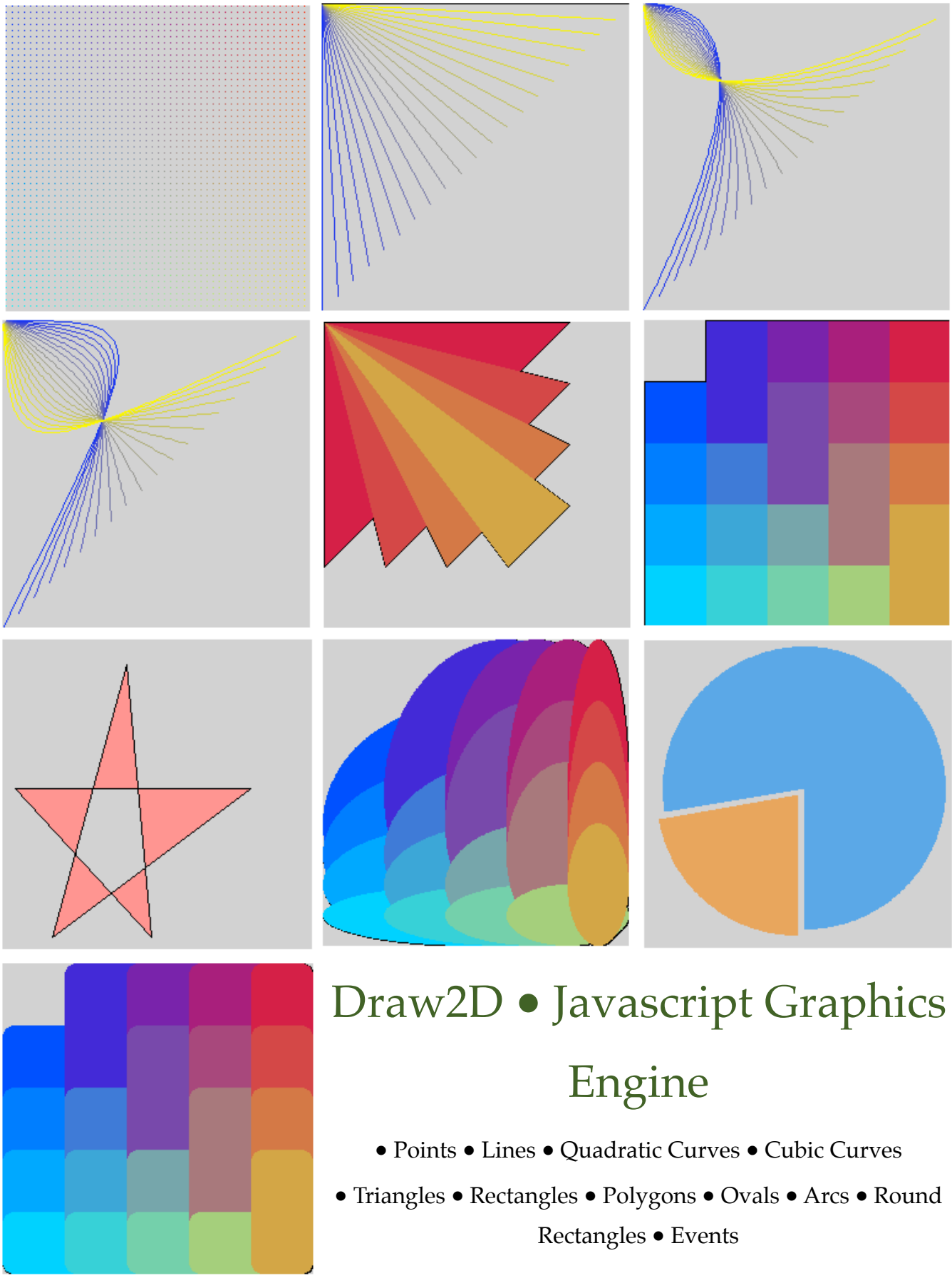


Draw2D • Javascript Graphics Engine

• Points • Lines • Quadratic Curves • Cubic Curves • Triangles • Rectangles • Polygons • Ovals • Arcs • Round Rectangles • Events

Draw2D • Javascript Graphics Engine

Updates

Version 1.1

Version 1.1 adds a step parameter to the constructors of curve drawing objects. This parameter can be be used to increase accuracy at the cost of speed or increase speed at the cost of accuracy. The new parameter is an optional parameter available as part of the constructors for the QuadraticCurve, CubicCurve, Oval, Arc, and RoundRect classes.

A bug with the rendering of CubicCurves was fixed.

Browsers

Compatibility

Draw2D uses only standard HTML DOM and should be compatible with any modern browser. It's been tested on the following browsers:


Safari.app


Firefox.app


Google Chrome.app


Opera.app

Speed

An informal benchmark on the performance of Draw2D was done on Safari, Firefox, Chrome, and Opera. The benchmark consisted of drawing and filling over 6000 rectangles.

Broadly, Safari and Chrome were fastest, while Firefox and Opera were noticeably slower.

For extra points, however, Opera has the nice feature of updating the window while the script runs, allowing you to watch the progress. All the other browsers wait until the script has competed before any window updates take place.

Versions

Compatibility and speed tests were done on a Macintosh using the following browser versions:

- Safari 4.0.5
- Firefox 3.0.18
- Chrome 5.0.307.11 beta
- Opera 10.10

Key Concepts

HTML And JavaScript

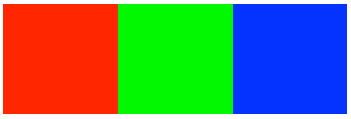
Draw2D is a JavaScript package that runs inside HTML pages, letting you add on the fly graphics to your pages. The **Draw2D** JavaScript code comes in a single file, Draw2d.js, which you'll include in your HTML file.

View

Think of a View as the piece of paper you'll use to draw your graphics. A View has a width, a height, and a collection of Shapes, such as Lines and Rectangles.

Inside your HTML page, you'll write JavaScript to create your View, create the Shapes you want on the View, and add any desired event handing.

Color



You'll use color to color the background of the View and the line drawing and fill colors used to draw Shapes. Colors have four components, red, green, blue, and alpha. The red, green, and blue values combine to specify a specific color. The alpha value specifies the transparency of the color.

Shapes

Draw2D provides several Shapes ready for you to use: Points, Lines, Quadratic Curves, Cubic Curves, Triangles, Rectangles, Polygons , Ovals, Arcs, and Round Rectangles. You'll create the Shapes you want to appear on the View, set their properties, including location, size, and color, and draw the View.

You can change the properties of a Shape, such as its color, during runtime if you wish, allowing the appearance of the Shape to dynamically change.

Shapes can have other Shapes as children, allowing you to work with a hierarchy of Shapes all at once.

Events

You can setup your Shapes to handle events. When the given event triggers, the function you supply will be executed. You can also set the colors used to draw the Shape when the event occurs.

Mouse click, mouse double click, mouse down, mouse up, mouse over, and mouse out events are supported.

Sample

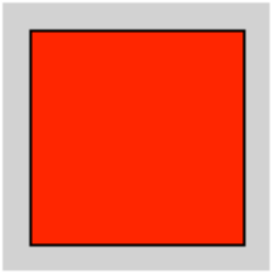
A Simple Example

This section provides a simple example of using **Draw2D**. Notable lines are highlighted in yellow. The script draws a rectangle inside an HTML page. The rectangle has a black border and a red interior and appears on a grey background.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title>New Web Project</title>
            <script type="text/javascript" src="./Draw2D.js"></script>
            <script type="text/javascript">
                function initDraw()
                {
                    var oGrey = new Color(200, 200, 200, 0);
                    var oBlack = new Color(0, 0, 0);
                    var oRed = new Color(255, 0, 0);
                    var oView = new View(0, 0, 100, 100, "View1",
document.body, document, oGrey);
                    var oRect = new Rect(10, 10, 80, 80, oBlack, oRed);

                    oView.addShape(oRect);
                    oView.draw(oView);
                    oView.fill(oView);
                } // initDraw
            </script>

    </head>
    <body onLoad="initDraw();">
    </body>
</html>
```



Notice a `Script` tag is used to include Draw2D.js. This is the **Draw2D** JavaScript code. Another `Script` tag is used to write a function called `initDraw()`. The `initDraw()` function creates 3 Colors, a View, and a Rectangle. The Rectangle is added to the View and the View is instructed to draw and fill all the Shapes it has.

Finally, the `Body` tag is set to call `initDraw()` when the HTML page starts to load. This sets all the drawing in motion. The end result is shown here.

HTML And JavaScript

HTML

HTML is a document that has several sections to it, and those sections can have sub-sections, which can also have sub-sections, and so on down the line. The sections are marked by tags, which are a collection of keywords the delimit the beginning and end of a section.

In the HTML example on the previous page, we can see an `HTML` tag, written as <html> , that marks the beginning of the HTML section (which is also called the Document section), along with a corresponding `</html>` tag at the end marking the end of the Document section. Within the Document section we can see tags for Header and Body sections marked with the `HEAD` and `BODY` tags. The Header and Body sections are subsections of the Document section.

Everything that appears on a web page is stored somewhere inside an HTML Document structure this way. Furthermore, the entire structure can be accessed using JavaScript.

The **Draw2D** script needs to know a bit about these sections so that it knows where in the HTML document to put the graphics it creates. In the example on the previous page,note the creation of a new View object in the middle of the HTML. There's a call to `new View()`, with several parameters being passed to `View()`. These parameters will be covered in detail in the View chapter, but for now, notice that `document` and `document.body` parameters.

The `document.body` parameter tells the **Draw2D** script where to place the items it creates. In this case, it says they will be a subsection below the Body section of the Document. This parameter doesn't necessarily have to be the Body section. It could be, for example, a Div section within the Body section. If you're familiar with what's known as the the JavaScript DOM (Document Object Model), you can walk the DOM to get any meaningful section you want and pass that section to the **Draw2D** script rather than using the Body section as was done in the sample.

The `document` parameter, on the other hand, always needs to be `document`, not anything else. The **Draw2D** script uses the `document` object to create the other objects it needs.

These are the only two parameters in **Draw2D** that reference the JavaScript DOM and in most cases you'll be able to use the values exactly as shown in the example on the previous page.

JavaScript

As its name implies, JavaScript is a very good scripting language. It can be used to quickly add short snippets of functionality to a web page. However, JavaScript is also a good object-oriented language. You can create objects that store both data and functions used to manipulate that data. You can also use a methodology known as *inheritance* that gives a JavaScript class access to all the data and methods of the class it inherits from. **Draw2D** uses JavaScript objects and inheritance.

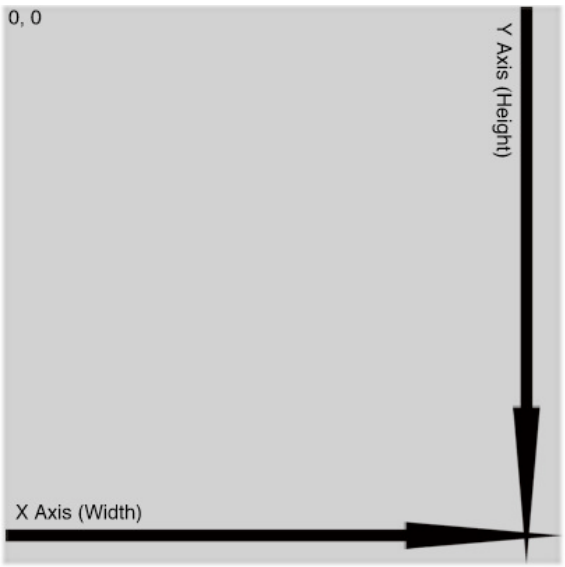
In the example on the previous page, the `new Color(), new View()` and `new Rect()` calls are examples of creating **Draw2D** JavaScript objects. These calls that create objects are known as *constructors*.

The `oView.addShape(oRect)` and `oView.draw(oView)` are examples of using the methods of an object.

Another feature of JavaScript is the support for optional parameters in function calls. Any parameter not specified is set to the value of `null`. **Draw2D** makes use of optional parameters to shorten the amount of information that needs to be passed to some constructors and functions. In these cases, Draw2D assigns reasonable default values for the missing parameters.

The description of a constructor or function that has optional parameters will indicate which parameters are optional and what the default values are.

View



Overview

A View acts as the paper upon which the Shapes of **Draw2D** are drawn. The View is nothing more than a rectangular box. This box has a coordinate system that specifies locations on the View, a background color that specifies the color of the View, and a collection of functions that can be called to pre-form operations on the view, such as adding and removing Shapes and drawing the View.

Coordinate System

The coordinate system has an X axis, which goes from left to right, and a Y axis which goes from top to bottom. Points along the axis are numbered, starting at 0. The coordinate of 0, 0 specifies the upper left corner of the View. Each point is sequentially numbered. The total number of points along the X axis is specified by the width of the View. The total number of points along the Y axis is specified by the Height of the View.

Shapes

Shapes are added and removed from a View using the `addShape()` and `removeShape()` functions. The `draw()` and `erase()` functions draw and erase the outlines of all contained Shapes. The `fill()` and `empty()` functions draw and erase the interiors of all contained Shapes.

Inheritance

The View class inherits from the Shape class.

Constructor

```
function View(inX, inY, inWidth, inHeight, inID, inParent, inDocument, inBackgroundColor)
```

PARAMETER	MEANING	OPTIONAL
`inX`	The X location of the upper left corner of the View.	No
`inY`	The Y location of the upper left corner of the View.	No
`inWidth`	The width of the View in pixels.	No
`inHeight`	The height of the View in pixels.	No
`inID`	A string containing the unique name of the View.	No

PARAMETER	MEANING	OPTIONAL
inParent	The JavaScript DOM object that is the parent of the View. This will often be set to document.body.	No
inDocument	The JAavaScript DOM object representing the document. Must be document.	No
inBackgroundColor	The background color of the view.	Yes. Defaults to transparent.

Sample

```
var oBackgroundColor = new Color(200, 200, 200, 0);
var oView = new View(0, 0, 100, 100, "View1", document.body, document,
oBackgroundColor);
```

Key Functions Defined In View

```
View.prototype.setBackgroundColor = function (inColor)
```

Sets the background color of the View.

PARAMETER	MEANING	OPTIONAL
inColor	The background color of the View.	No

Color

Red, Green, Blue, Alpha, And Transparent

Colors are defined using 3 numeric values between 0 and 255. A fourth value, called alpha, specifies the opacity of the color and has a range of 0 to 1. These 4 values represent the amount of red, green, and blue that make of the color, and the opacity of the color. Another way to specify a color is to simply declare it to be transparent. A transparent color means no color at all and to use whatever colors are behind the object being declared transparent. Both the alpha value and the transparent declaration are optional when creating a color.

Lower numbers means less of that color should be used to make up the final color. Higher values mean more of that color should be used.

Inheritance

None.

Constructor

```
function Color(inRed, inGreen, inBlue, inAlpha, inTransparent)
```

PARAMETER	MEANING	OPTIONAL
`inRed`	The amount of red in the final color. Valid values are integers between 0 and 255.	No
`inGreen`	The amount of green in the final color. Valid values are integers between 0 and 255.	No
`inBlue`	The amount of blue in the final color. Valid values are integers between 0 and 255.	No
`inAlpha`	The amount of opacity in the color. Valid values are real numbers between 0 and 1.	Yes. Defaults to 1.0 (a solid color)
`inTransparent`	True if the color is transparent (completely invisible), false otherwise.	Yes. Defaults to `false`.

Samples

```
// White
new Color(255, 255, 255);
// Black
new Color(0, 0, 0);
// Medium Grey
new Color(127, 127, 127);
// Dark Grey
new Color(64, 64, 64);
// Light Grey
new Color(191, 191, 191);
// Red
new Color(255, 0, 0);
// Green
new Color(0, 255, 0);
// Blue
new Color(0, 0, 255);
// Cyan
new Color(0, 255, 255);
// Purple
new Color(255, 0, 255);
// Yellow
new Color(255, 255, 0);
// Black With 1/2 Opacity (See-Thru)
new Color(0, 0, 0, 0.5);
// Fully Transparent (Invisible)
new Color(0, 0, 0, 0, true);
```

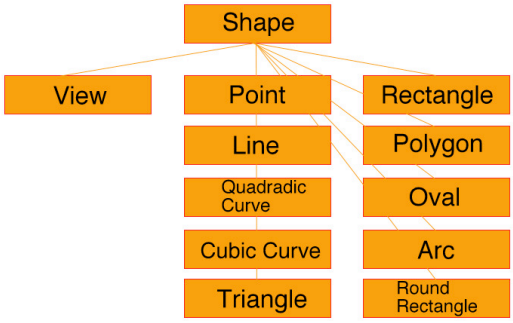
Key Functions Defined In Color

`Color.prototype.isEqual = function (inColor)`
Returns `true` if this color object represents the same color as another color object, `false` otherwise.

PARAMETER	MEANING	OPTIONAL
`inColor`	The Color object being compared to this Color object.	No

```
Color.prototype.invert = function()
```
Inverts the color represented by this object.

Shapes



Shape Classes

Shapes classes encapsulate nearly all of the functionality of the Draw2D package. The base class for all shapes is the Shape class. The other classes, View, Point, Rectangle, and so on, inherit from Shape and therefore share all its capabilities. A diagram of the Shape inheritance hierarchy is shown to the left.

Shape Base Class

Shapes are the base class for all other shape classes. You shouldn't ever create a Shape object directly, but knowing the functionality Shape provides is important because this functionality is inherited by all other shape classes. The following key functionality is define in the Shape base class:

• Shapes can be added and removed to other shapes to for parent/child relationships. Usually this is used to add and remove shapes to and from a View.

• Shapes have an area.

• Shapes can be checked to see if a given point lies within the shape.

• Shapes can be checked for equality. Two shapes are equal if their X, Y, width, and height values are equal and all their children are equal.

• Shapes can be drawn and erased, which displays and hides the outline of a shape. Shapes have a draw color.

• Shapes can be filled and emptied, which displays and hides the interior of a shape. Shapes have a fill color.

• The children of a shape can be drawn, erased, filled, and emptied separate from the shape itself.

• Shapes can be assigned event information to handle HTML events.

Shape Inheritance

None.

Shape Constructor

```
function Shape()
```

Key Functions Defined In Shape

```
Shape.prototype.addShape = function (inShape)
```
Adds a shape object as a child to this shape.

PARAMETER	MEANING	OPTIONAL
`inShape`	The shape object to add as a child.	No

`Shape.prototype.removeShape = function (inShape)`
Removes a shape object as a child from this shape.

PARAMETER	MEANING	OPTIONAL
`inShape`	The shape object to remove as a child.	No

`Shape.prototype.getArea = function ()`
Returns the area of the shape, as measuring in pixels.

`Shape.prototype.containsPoint = function (inPoint)`
Returns true if the shape contains the given point, false otherwise.

PARAMETER	MEANING	OPTIONAL
`inPoint`	The Point object to check.	No

`Shape.prototype.isEqual = function (inShape)`
Returns true if the shape is equal to the given shape, false otherwise. Two shapes are equal if their X, Y, width, and height values are equal and all their children are equal.

PARAMETER	MEANING	OPTIONAL
`inShape`	The shape object to compare for equality.	No

`Shape.prototype.draw = function (inView)`
Draws the outline of the shape on the given View.

PARAMETER	MEANING	OPTIONAL
`inView`	The View to draw the shape on.	No

`Shape.prototype.drawChildren = function (inView)`
Draws the outline of the children of the shape on the given View.

PARAMETER	MEANING	OPTIONAL
`inView`	The View to draw the children of the shape on.	No

`Shape.prototype.erase = function (inView)`
Erases the outline of the shape from the given View.

PARAMETER	MEANING	OPTIONAL
`inView`	The View to erase the shape from.	No

`Shape.prototype.eraseChildren = function (inView)`
Erases the outline of the children of the shape from the given View.

PARAMETER	MEANING	OPTIONAL
`inView`	The View to erase the children of the shape from.	No

`Shape.prototype.fill = function (inView, inEraseFlag)`
Draws the interior of the shape on the given View.

PARAMETER	MEANING	OPTIONAL
`inView`	The View to fill the shape to.	No

`Shape.prototype.fillChildren = function (inView)`
Draws the interior of the children of the shape on the given View.

PARAMETER	MEANING	OPTIONAL
`inView`	The View to fill the children of the shape on.	No

`Shape.prototype.empty = function (inView)`
Erases the interior of the shape from the given View.

PARAMETER	MEANING	OPTIONAL
`inView`	The View to empty the shape from.	No

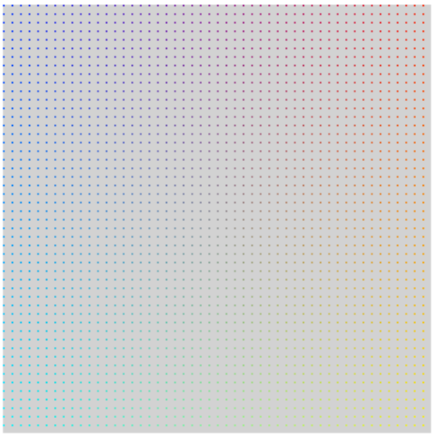
`Shape.prototype.emptyChildren = function (inView)`
Erases the interior of the children of the shape from the given View.

PARAMETER	MEANING	OPTIONAL
`inView`	The View to empty the children of the shape from.	No

`Shape.prototype.setEventInfo = function (inEventInfo)`

Sets the event handling information for the shape.

PARAMETER	MEANING	OPTIONAL
`inEventInfo`	An EventInfo object containing the event handling information for the shape.	No



Point Class

The Point class is used for drawing points and for performing a few basic mathematical operations on points.

Point Inheritance

The Point class inherits from the Shape class.

Point Constructor

```
function Point(inX, inY, inDrawColor)
```

PARAMETER	MEANING	OPTIONAL
`inX`	The location of the Point on the X axis of the parent View.	No
`inY`	The location of the Point on the Y axis of the parent View.	No
`inDrawColor`	The Color object used to draw the point.	No

Key Functions Defined In Point

`Point.prototype.add = function (inPoint)`

Mathematically adds the X and Y values of the inPoint parameter to the X and Y values of this point. The result is stored in a new Point object which is returned as the value of the function.

PARAMETER	MEANING	OPTIONAL
`inPoint`	A Point object.	No

`Point.prototype.subtract = function (inPoint)`

Mathematically subtracts the X and Y values of the inPoint parameter from the X and Y values of this point. The result is stored in a new Point object which is returned as the value of the function.

PARAMETER	MEANING	OPTIONAL
inPoint	A Point object.	No

Point.prototype.getDistanceSquared = function (inPoint)

Returns the squared distance between this Point and the passed in Point object.

PARAMETER	MEANING	OPTIONAL
inPoint	A Point object.	No

Point.prototype.average = function(inPoint)

Mathematically calculates the average of this Point and the inPoint parameter by adding the two together and multiplying the resulting X and Y values by 0.5. The result is stored in a new Point object which is returned as the value of the function.

PARAMETER	MEANING	OPTIONAL
inPoint	A Point object.	No

Point.prototype.dotProduct = function(inPoint)

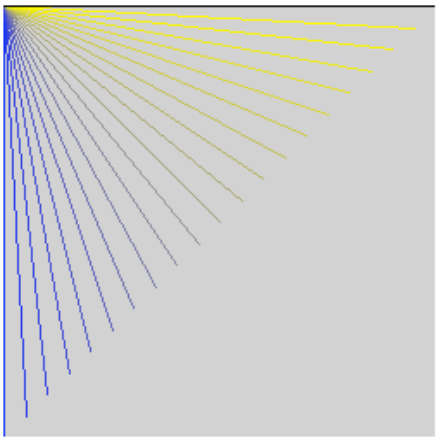
Returns the dot product of this Point object and the inPoint parameter. The dot product is a floating point number, not a Point object.

PARAMETER	MEANING	OPTIONAL
inPoint	A Point object.	No

Point.prototype.crossProduct = function(inPoint)

Returns the cross product of this Point object and the inPoint parameter. The cross product is a floating point number, not a Point object.

PARAMETER	MEANING	OPTIONAL
inPoint	A Point object.	No



Line Class

The Line class is used for drawing lines and for performing a few basic mathematical operations on lines.

Line Inheritance

The Line class inherits from the Shape class.

Line Constructor

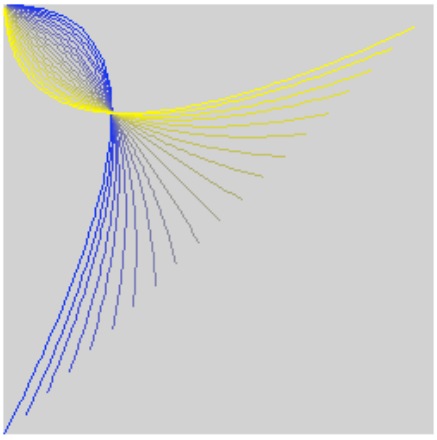
```
function Line(inStartPoint, inEndPoint, in-
DrawColor)
```

PARAMETER	MEANING	OPTIONAL
`inStartPoint`	A Point object defining the line's starting location in the parent View.	No
`inEndPoint`	A Point object defining the line's ending location in the parent View.	No
`inDrawColor`	The Color object used to draw the line.	No

Key Functions Defined In Line

```
Line.prototype.getLength = function ()
```
Returns the length of the line. Note that this function performs a square root operation, which can be slow.

```
Line.prototype.getLengthSquared = function ()
```
Returns the squared length of the line.



QuadraticCurve Class

The QuadraticCurve class is used for quadratic curves. Quadratic curves have a start point an end point and a center point. The distance of the center point from the start and end point define the shape of the curve.

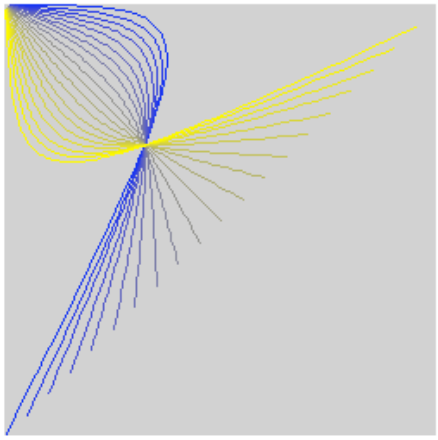
QuadraticCurve Inheritance

The QuadraticCurve class inherits from the Shape class.

QuadraticCurve Constructor

```
function QuadraticCurve(inEndPoint1, inEnd-
Point2, inCenterPoint, inDrawColor, in-
Steps)
```

PARAMETER	MEANING	OPTIONAL
`inStartPoint`	A Point object defining the curve's starting location in the parent View.	No
`inEndPoint`	A Point object defining the curve's ending location in the parent View.	No
`inCenterPoint`	A Point object defining the curve's center location in the parent View.	No
`inDrawColor`	The Color object used to draw the curve.	No
`InSteps`	Controls the number of steps used to draw the curve. Higher numbers provide a more accurate rendering at the cost of speed. Lower numbers sacrifice accuracy for improved speed.	Yes. Defaults to 100.



CubicCurve Class

The CubicCurve class is used for cubic curves. Cubic curves have a start point an end point and two center point. The distance of the center points from the start and end point define the shape of the curve.

CubicCurve Inheritance

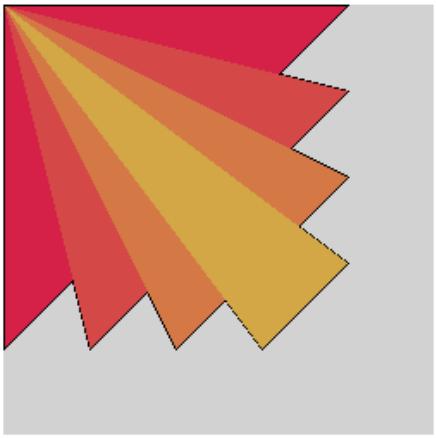
The CubicCurve class inherits from the Shape class.

CubicCurve Constructor

```
function CubicCurve(inEndPoint1, inEnd-
Point2, inCenterPoint1, inCenterPoint2, in-
DrawColor, inSteps)
```

PARAMETER	MEANING	OPTIONAL
`inStartPoint`	A Point object defining the curve's starting location in the parent View.	No
`inEndPoint`	A Point object defining the curve's ending location in the parent View.	No
`inCenterPoint1`	A Point object defining the curve's center location in the parent View.	No
`inCenterPoint2`	A Point object defining the curve's center location in the parent View.	No

PARAMETER	MEANING	OPTIONAL
`inDrawColor`	The Color object used to draw the curve.	No
`inSteps`	Controls the number of steps used to draw the curve. Higher numbers provide a more accurate rendering at the cost of speed. Lower numbers sacrifice accuracy for improved speed.	Yes. Defaults to 100.



Triangle Class

The Triangle class is used for triangles. Triangles are defined by three different points on a View, a draw color used to render the outline of the Triangle and a fill color used to render the interior of the Triangle.

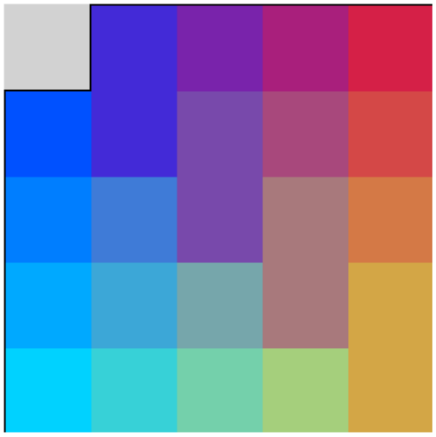
Triangle Inheritance

The Triangle class inherits from the Shape class.

Triangle Constructor

```
function Triangle(inPoint1, inPoint2, inPoint3, inDrawColor, inFillColor)
```

PARAMETER	MEANING	OPTIONAL
`inPoint1`	One of three point Objects used to define the location of the Triangle in the parent View.	No
`inPoint2`	One of three point Objects used to define the location of the Triangle in the parent View.	No
`inPoint3`	One of three point Objects used to define the location of the Triangle in the parent View.	No
`inDrawColor`	The Color object used to draw the outline of the Triangle.	No
`inFillColor`	The Color object used to draw the interior of the Triangle.	No



Rect Class

The Rect class is used for rectangles. Rectangles are defined by an X and Y location of the upper left corner of the rectangle, the width and height of the rectangle, a draw color used to render the outline of the rectangle and a fill color used to render the interior of the rectangle.

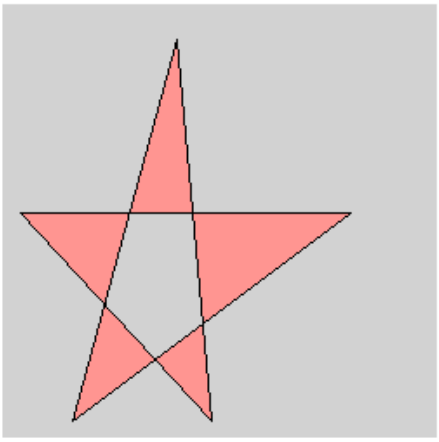
Rect Inheritance

The Rect class inherits from the Shape class.

Rect Constructor

```
function Rect(inX, inY, inWidth, inHeight, inDrawColor, inFillColor)
```

PARAMETER	MEANING	OPTIONAL
`inX`	The location of the upper left corner of the Rectangle on the X axis of the parent View.	No
`inY`	The location of the upper left corner of the Rectangle on the Y axis of the parent View.	No
`inWidth`	The width of the rectangle.	No
`inHeight`	The height of the rectangle.	No
`inDrawColor`	The Color object used to draw the outline of the Rectangle.	No
`inFillColor`	The Color object used to draw the interior of the Rectangle.	No



Polygon Class

The Polygon class is used to define polygons. Polygons are made up of a collection of sequential points, a flag indicating if the last point in the list should be connected to the first, a Color used to render the outline of the Polygon, and a Color used to render the interior of the Polygon.

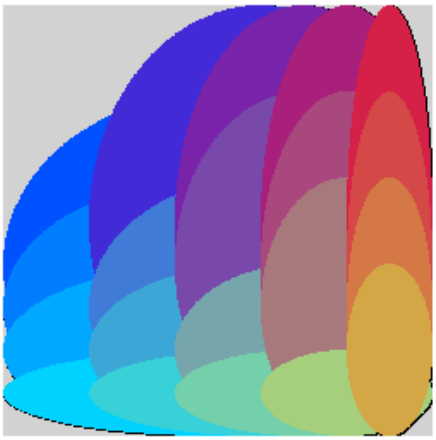
Polygon Inheritance

The Polygon class inherits from the Shape class.

Polygon Constructor

```
function Polygon(inPoints, inCloseLoop, inDrawColor, inFillColor)
```

PARAMETER	MEANING	OPTIONAL
`inPoints`	A JavaScript array containing Point objects that define the boundary of the Polygon.	No
`inCloseLoop`	Set to true if the last Point of the Polygon should be connected to the first Point when rendering the boundary of the Polygon.	No
`inDrawColor`	The Color object used to draw the outline of the Rectangle.	No
`inFillColor`	The Color object used to draw the interior of the Rectangle.	No



Oval Class

The Oval class is used to define ovals. Ovals are defined by an X and Y location of the upper left corner of the oval, the width and height of the oval, a draw color used to render the outline of the oval and a fill color used to render the interior of the oval.

An oval with an equal width and height is a circle.

Oval Inheritance

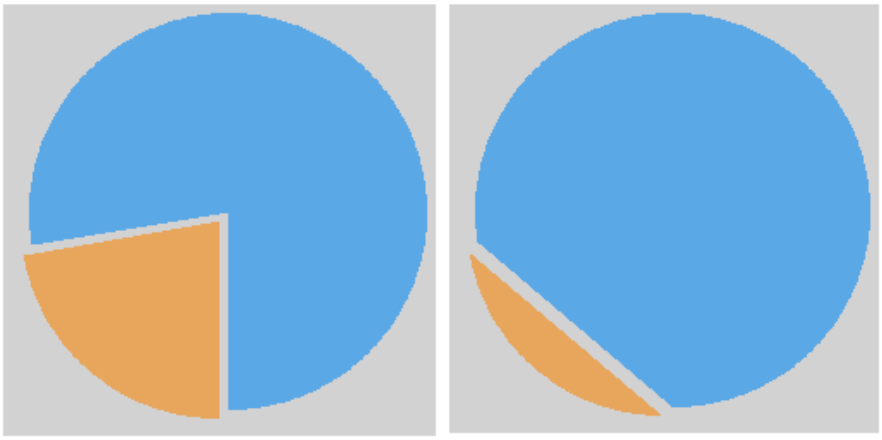
The Oval class inherits from the Shape class.

Oval Constructor

```
function Oval(inX, inY, inWidth, inHeight,
inDrawColor, inFillColor, inStep)
```

PARAMETER	MEANING	OPTIONAL
`inX`	The location of the upper left corner of the Oval on the X axis of the parent View.	No
`inY`	The location of the upper left corner of the Oval on the Y axis of the parent View.	No
`inWidth`	The width of the Oval.	No
`inHeight`	The height of the Oval.	No
`inDrawColor`	The Color object used to draw the outline of the Oval.	No
`inFillColor`	The Color object used to draw the interior of the Oval.	No

PARAMETER	MEANING	OPTIONAL
`inStep`	Controls how many lines are used to draw curves. Smaller numbers give more accurate results at the cost of speed. Larger numbers give faster results at the cost of accuracy.	Yes. Defaults to 0.01.



Arc Class

The Arc class is used to define arcs. Arc are defined by an X and Y location of the upper left corner of the arc, the width and height of the arc, the start and end angle of the arc, whether or not the arc should connect to the center of the circle on which it is formed, a draw color used to render the outline of the arc and a fill color used to render the interior of the arc.

Arc Inheritance

The Arc class inherits from the Shape class.

Arc Constructor

```
function Arc(inX, inY, inWidth, inHeight, inStartAngle, inEndAngle,
inConnectToCenter, inDrawColor, inFillColor, inStep)
```

PARAMETER	MEANING	OPTIONAL
`inX`	The location of the upper left corner of the Arc on the X axis of the parent View.	No
`inY`	The location of the upper left corner of the Arc on the Y axis of the parent View.	No
`inWidth`	The width of the Arc.	No
`inHeight`	The height of the Arc.	No
`inStartAngle`	The angle on the circle where the Arc begins. Angle zero is at the bottom of the circle. Increased angles move counter-clockwise.	No

PARAMETER	MEANING	OPTIONAL
`inEndAngle`	The angle on the circle where the Arc ends. Angle zero is at the bottom of the circle. Increased angles move counter-clockwise.	No
`inConnectToCenter`	A boolean value indicating if the arc should connect to the center of the circle it's on, as in a pie chart, or simply have the end of the arc connect to the beginning of the arc with a single straight line.	No
`inDrawColor`	The Color object used to draw the outline of the Arc.	No
`inFillColor`	The Color object used to draw the interior of the Arc.	No
`inStep`	Controls how many lines are used to draw curves. Smaller numbers give more accurate results at the cost of speed. Larger numbers give faster results at the cost of accuracy.	Yes. Defaults to 0.01.

RoundRect Class

The RoundRect class is used to define a rectangle with rounded corners. RoundRect are defined by an X and Y location of the upper left corner of the RoundRect, the width and height of the RoundRect, the width and height of the Oval used to draw the rounded corners of the RoundRect, a draw color used to render the outline of the RoundRect and a fill color used to render the interior of the RoundRect.

RoundRect Inheritance

The RoundRect class inherits from the Shape class.

RoundRect Constructor

```
function RoundRect(inX, inY, inWidth, inHeight, inOvalWidth, inOval-
Height, inDrawColor, inFillColor, inStep)
```

PARAMETER	MEANING	OPTIONAL
`inX`	The location of the upper left corner of the RoundRect on the X axis of the parent View.	No
`inY`	The location of the upper left corner of the RoundRect on the Y axis of the parent View.	No
`inWidth`	The width of the RoundRect.	No
`inHeight`	The height of the RoundRect.	No
`inOvalWidth`	The width of the Oval used to draw the RoundRect's corners.	No
`inOvalHeight`	The height of the Oval used to draw the RoundRect's corners.	No
`inDrawColor`	The Color object used to draw the outline of the RoundRect.	No
`inFillColor`	The Color object used to draw the interior of the RoundRect.	No
`inStep`	Controls how many lines are used to draw curves. Smaller numbers give more accurate results at the cost of speed. Larger numbers give faster results at the cost of accuracy.	Yes. Defaults to 0.01.

Events

Events

The shapes of Draw2D can be programmed to respond to various mouse events. You can program the shapes to execute a function or change their draw color or fill color when a supported mouse event occurs. The mouse events supported are:

- Mouse Click
- Mouse Down
- Mouse Up
- Mouse Double Click
- Mouse Over
- Mouse Out

EventInfo Class

The EventInfo class stores information about how to handle events. The class has two data members, mouseClickInfo and mouseMoveInfo. You can set these using the EventInfo constructor, or at any time after construction. You then use the setEventInfo() function of the Shape class to assign the EventInfo object to a Shape and give it event handling capabilities.

EventInfo Inheritance

None.

EventInfo Constructor

```
function EventInfo(inMouseClickInfo, inMouseMoveInfo)
```

PARAMETER	MEANING	OPTIONAL
`inMouseClickInfo`	A MouseClickInfo object defining how the Shape handles mouse click events.	Yes
`inMouseMoveInfo`	A MouseMoveInfo object defining how the Shape handles mouse move events.	Yes

- Mouse Out

MouseClickInfo Class

The MouseClickInfo class stores information about how to handle mouse click events. Members of the class provide the ability to set a function, a draw color, and a fill color that are used whenever any of the following events occur.

- Mouse Click
- Mouse Down
- Mouse Up
- Mouse Double Click

Each of these events can have unique functions, draw colors, and fill colors.

MouseClickInfo Inheritance

None.

MouseClickInfo Constructor

```
function MouseClickInfo(inMouseClickFunction, inMouseClickColor, inMouseClickFillColor, inMouseDownFunction, inMouseDownColor, inMouseDownFillColor, inMouseUpFunction, inMouseUpColor, inMouseUpFillColor, inMouseDoubleClickFunction, inMouseDoubleClickColor, inMouseDoubleClickFillColor)
```

PARAMETER	MEANING	OPTIONAL
`inMouseClickFunction`	The function called to handle mouse click events that occur on the Shape.	Yes
`inMouseClickColor`	The Color used to draw the Shape when mouse click events occur.	Yes
`inMouseClickFillColor`	The Color used to fill the Shape when mouse click events occur.	Yes
`inMouseDownFunction`	The function called to handle mouse down events that occur on the Shape.	Yes
`inMouseDownColor`	The Color used to draw the Shape when mouse down events occur.	Yes
`inMouseDownFillColor`	The Color used to fill the Shape when mouse down events occur.	Yes
`inMouseUpFunction`	The function called to handle mouse up events that occur on the Shape.	Yes
`inMouseUpColor`	The Color used to draw the Shape when mouse up events occur.	Yes
`inMouseUpFillColor`	The Color used to fill the Shape when mouse up events occur.	Yes

PARAMETER	MEANING	OPTIONAL
`inMouseDoubleClickFunction`	The function called to handle mouse double click events that occur on the Shape.	Yes
`inMouseDoubleClickColor`	The Color used to draw the Shape when mouse double click events occur.	Yes
`inMouseDoubleClickFillColor`	The Color used to fill the Shape when mouse double click events occur.	Yes

The MouseClickInfo class has the following members, all of which can be set directly in the code.

```
this.mouseClickFunction
this.mouseClickColor
this.mouseClickFillColor
this.mouseDownFunction
this.mouseDownColor
this.mouseDownFillColor
this.mouseUpFunction
this.mouseUpColor
this.mouseUpFillColor
this.mouseDoubleClickFunction
this.mouseDoubleClickColor
this.mouseDoubleClickFillColor
```

MouseMoveInfo Class

The MouseMoveInfo class stores information about how to handle mouse move events. Members of the class provide the ability to set a function, a draw color, and a fill color that are used whenever any of the following events occur.

- Mouse Over
- Mouse Out

Each of these events can have unique functions, draw colors, and fill colors.

MouseMoveInfo Inheritance

None.

MouseMoveInfo Constructor

```
function MouseMoveInfo(inMouseOverFunction, inMouseOverColor, inMouseOverFillColor, inMouseOutFunction, inMouseOutColor, inMouseOutFillColor)
```

PARAMETER	MEANING	OPTIONAL
`inMouseOverFunction`	The function called to handle mouse click over that occur on the Shape.	Yes
`inMouseOverColor`	The Color used to draw the Shape when mouse over events occur.	Yes
`inMouseOverFillColor`	The Color used to fill the Shape when mouse over events occur.	Yes
`inMouseOutFunction`	The function called to handle mouse over events that occur on the Shape.	Yes
`inMouseOutColor`	The Color used to draw the Shape when mouse over events occur.	Yes
`inMouseOutFillColor`	The Color used to fill the Shape when mouse over events occur.	Yes

The MouseMoveInfo class has the following members, all of which can be set directly in the code.

```
this.mouseOverFunction
this.mouseOverColor
this.mouseOverFillColor
this.mouseOutFunction
this.mouseOutColor
this.mouseOutFillColor
```

Event Functions

The event function used to handle events and assigned to MouseClickInfo and mouseMoveInfo members is a simpe function that is passed no parameters and its return value is never checked. The code below shows a simple event function.

```
var oMouseMoveInfo = new MouseMoveInfo();

oMouseMoveInfo.mouseOverFunction = function(){};
```

The Code example below hows how to assign event handling code to a Shape.

```
var oRect = new Rect(10, 10, 80, 80, oBlack, oRed);
var oMouseClickInfo = new MouseClickInfo();
var oMouseMoveInfo = new MouseMoveInfo();
var oEventInfo = new EventInfo();
var oBlack = new Color(0, 0, 0);
var oRed = new Color(255, 0, 0);
var oGreen = new Color(0, 255, 0);
var oBlue = new Color(0, 0, 255);
var oYellow = new Color(255, 255, 0);
var oPurple = new Color(255, 0, 255);

oEventInfo.mouseClickInfo = oMouseClickInfo;
oEventInfo.mouseMoveInfo = oMouseMoveInfo;
oMouseMoveInfo.mouseOverFunction = function(){};
oMouseMoveInfo.mouseOverColor = oBlue;
oMouseMoveInfo.mouseOverFillColor = oBlue;
oMouseMoveInfo.mouseOutFunction = function(){};
oMouseMoveInfo.mouseOutColor = oBlack;
oMouseMoveInfo.mouseOutFillColor = oRed;
oMouseClickInfo.mouseClickFunction = function(){alert("click");};
oMouseClickInfo.mouseClickColor = oYellow;
oMouseClickInfo.mouseClickFillColor = oYellow;
oMouseClickInfo.mouseDoubleClickFunction = function(){alert("double
click");};
oMouseClickInfo.mouseDoubleClickColor = oGreen;
oMouseClickInfo.mouseDoubleClickFillColor = oGreen;
oMouseClickInfo.mouseDownFunction = function(){alert("down");};
oMouseClickInfo.mouseDownColor = oPurple;
oMouseClickInfo.mouseDownFillColor = oPurple;
oMouseClickInfo.mouseUpFunction = function(){alert("up");};
oMouseClickInfo.mouseUpColor = oBlack;
oMouseClickInfo.mouseUpFillColor = oRed;
oRect.setEventInfo(oEventInfo);
```