UIBezierPath Class Reference

Graphics & Animation

2010-05-20

Contents

Figures

UIBezierPath Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/UIKit.framework
Availability	Available in iOS 3.2 and later.
Companion guide	iPad Programming Guide
Declared in	UIBezierPath.h

Overview

The `UIBezierPath` class lets you define a path consisting of straight and curved line segments and render that path in your custom views. You use this class initially to specify just the geometry for your path. Paths can define simple shapes such as rectangles, ovals, and arcs or they can define complex polygons that incorporate a mixture of straight and curved line segments. After defining the shape, you can use additional methods of this class to render the path in the current drawing context.

A `UIBezierPath` object combines the geometry of a path with attributes that describe the path during rendering. You set the geometry and attributes separately and can change them independent of one another. Once you have the object configured the way you want it, you can tell it to draw itself in the current context. Because the creation, configuration, and rendering process are all distinct steps, Bezier path objects can be reused easily in your code. You can even use the same object to render the same shape multiple times, perhaps changing the rendering options between successive drawing calls.

You set the geometry of a path by manipulating the path's current point. When you create a new empty path object, the current point is undefined and must be set explicitly. To move the current point without drawing a segment, you use the `moveToPoint:` (page 25) method. All other methods result in the addition of either a line or curve segments to the path. The methods for adding new segments always assume you are starting at the current point and ending at some new point that you specify. After adding the segment, the end point of the new segment automatically becomes the current point.

A single Bezier path object can contain any number of open or closed subpaths, where each subpath represents a connected series of path segments. Calling the `closePath` (page 22) method closes a subpath by adding a straight line segment from the current point to the first point in the subpath. Calling the `moveToPoint:` method ends the current subpath (without closing it) and sets the starting point of the next subpath. The subpaths of a Bezier path object share the same drawing attributes and must be manipulated as a group. To draw subpaths with different attributes, you must put each subpath in its own `UIBezierPath` object.

After configuring the geometry and attributes of a Bezier path, you draw the path in the current graphics context using the `stroke` (page 26) and `fill` (page 23) methods. The `stroke` method traces the outline of the path using the current stroke color and the attributes of the Bezier path object. Similarly, the `fill` method fills in the area enclosed by the path using the current fill color. (You set the stroke and fill color using the `UIColor` class.)

In addition to using a Bezier path object to draw shapes, you can also use it to define a new clipping region. The `addClip` (page 18) method intersects the shape represented by the path object with the current clipping region of the graphics context. During subsequent drawing, only content that lies within the new intersection region is actually rendered to the graphics context.

Tasks

Creating a UIBezierPath Object

`+ bezierPath` (page 14)
    Creates and returns a new `UIBezierPath` object.

`+ bezierPathWithRect:` (page 16)
    Creates and returns a new `UIBezierPath` object initialized with a rectangular path.

`+ bezierPathWithOvalInRect:` (page 16)
    Creates and returns a new `UIBezierPath` object initialized with an oval path inscribed in the specified rectangle

`+ bezierPathWithRoundedRect:cornerRadius:` (page 17)
    Creates and returns a new `UIBezierPath` object initialized with a rounded rectangular path.

`+ bezierPathWithRoundedRect:byRoundingCorners:cornerRadii:` (page 17)
    Creates and returns a new `UIBezierPath` object initialized with a rounded rectangular path.

`+ bezierPathWithArcCenter:radius:startAngle:endAngle:clockwise:` (page 14)
    Creates and returns a new `UIBezierPath` object initialized with an arc of a circle.

`+ bezierPathWithCGPath:` (page 15)
    Creates and returns a new `UIBezierPath` object initialized with the contents of a Core Graphics path.

Constructing a Path

`- moveToPoint:` (page 25)
    Moves the receiver's current point to the specified location.

`- addLineToPoint:` (page 20)
    Appends a straight line to the receiver's path.

`- addArcWithCenter:radius:startAngle:endAngle:clockwise:` (page 18)
    Appends an arc to the receiver's path.

`- addCurveToPoint:controlPoint1:controlPoint2:` (page 19)
    Appends a cubic Bézier curve to the receiver's path.

`- addQuadCurveToPoint:controlPoint:` (page 21)
    Appends a quadratic Bézier curve to the receiver's path.

- `closePath` (page 22)
  Closes the most recently added subpath.
- `removeAllPoints` (page 25)
  Removes all points from the receiver, effectively deleting all subpaths.
- `appendPath:` (page 21)
  Appends the contents of the specified path object to the receiver's path.

`CGPath` (page 10) *property*
  The Core Graphics representation of the path.

`currentPoint` (page 11) *property*
  The current point in the graphics path. (read-only)

Accessing Drawing Properties

`lineWidth` (page 13) *property*
  The line width of the path.

`lineCapStyle` (page 12) *property*
  The shape of the paths end points when stroked.

`lineJoinStyle` (page 12) *property*
  The shape of the joints between connected segments of a stroked path.

`miterLimit` (page 13) *property*
  The limiting value that helps avoid spikes at junctions between connected line segments.

`flatness` (page 12) *property*
  The factor that determines the rendering accuracy for curved path segments.

`usesEvenOddFillRule` (page 13) *property*
  A Boolean indicating whether the even-odd winding rule is in use for drawing paths.

- `setLineDash:count:phase:` (page 25)
  Sets the line-stroking pattern for the path.
- `getLineDash:count:phase:` (page 24)
  Retrieves the line-stroking pattern for the path.

Drawing Paths

- `fill` (page 23)
  Paints the region enclosed by the receiver's path using the current drawing properties.
- `fillWithBlendMode:alpha:` (page 24)
  Paints the region enclosed by the receiver's path using the specified blend mode and transparency values.
- `stroke` (page 26)
  Draws a line along the receiver's path using the current drawing properties.
- `strokeWithBlendMode:alpha:` (page 26)
  Draws a line along the receiver's path using the specified blend mode and transparency values.

Clipping Paths

`– addClip` (page 18)
: Intersects the area enclosed by the receiver's path with the clipping path of the current graphics context and makes the resulting shape the current clipping path.

Hit Detection

`– containsPoint:` (page 23)
: Returns a Boolean value indicating whether the area enclosed by the receiver contains the specified point.

`empty` (page 11) *property*
: A Boolean value indicating whether the path has any valid elements. (read-only)

`bounds` (page 10) *property*
: The bounding rectangle of the path. (read-only)

Applying Transformations

`– applyTransform:` (page 22)
: Transforms all points in the path using the specified affine transform matrix.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

bounds

The bounding rectangle of the path. (read-only)

```
@property(nonatomic, readonly) CGRect bounds
```

Discussion
The value in this property represents the smallest rectangle that completely encloses all points in the path, including any control points for Bézier and quadratic curves.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

CGPath

The Core Graphics representation of the path.

```
@property(nonatomic) CGPathRef CGPath
```

Discussion
This property contains a snapshot of the path at any given point in time. Getting this property returns an immutable path object that you can pass to Core Graphics functions. The path object itself is owned by the `UIBezierPath` object and is valid only until you make further modifications to the path.

You can set the value of this property to a path you built using the functions of the Core Graphics framework. When setting a new path, this method makes a copy of the path you provide.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

currentPoint

The current point in the graphics path. (read-only)

```
@property(nonatomic, readonly) CGPoint currentPoint
```

Discussion
The value in this property represents the starting point for new line and curve segments. If the path is currently empty, this property contains the value `CGPointZero`.

Availability
Available in iOS 3.2 and later.

See Also
  `@property empty` (page 11)

Declared In
`UIBezierPath.h`

empty

A Boolean value indicating whether the path has any valid elements. (read-only)

```
@property(readonly, getter=isEmpty) BOOL empty
```

Discussion
Valid path elements include commands to move to a specified point, draw a line or curve segment, or close the path. Thus, a path is not considered empty even if all you do is call the `moveToPoint:` (page 25) method.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

flatness

The factor that determines the rendering accuracy for curved path segments.

```
@property(nonatomic) CGFloat flatness
```

Discussion
The flatness value measures the largest permissible distance (measured in pixels) between a point on the true curve and a point on the rendered curve. Smaller values result in smoother curves but require more computation time. Larger values result in more jagged curves but are rendered much faster. The default flatness value is `0.6`.

In most cases, you should not change the flatness value. However, you might increase the flatness value temporarily to minimize the amount of time it takes to draw a shape temporarily (such as during scrolling).

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

lineCapStyle

The shape of the paths end points when stroked.

```
@property(nonatomic) CGLineCap lineCapStyle
```

Discussion
The line cap style is applied to the start and end points of any open subpaths. This property does not affect closed subpaths. The default line cap style is `kCGLineCapButt`.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

lineJoinStyle

The shape of the joints between connected segments of a stroked path.

```
@property(nonatomic) CGLineJoin lineJoinStyle
```

Discussion
The default line join style is `kCGLineJoinMiter`.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

lineWidth

The line width of the path.

```
@property(nonatomic) CGFloat lineWidth
```

Discussion
The line width defines the thickness of the receiver's stroked path. A width of 0 is interpreted as the thinnest line that can be rendered on a particular device. The actual rendered line width may vary from the specified width by as much as 2 device pixels, depending on the position of the line with respect to the pixel grid and the current anti-aliasing settings. The width of the line may also be affected by scaling factors specified in the current transformation matrix of the active graphics context.

The default line width is 1.0.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

miterLimit

The limiting value that helps avoid spikes at junctions between connected line segments.

```
@property(nonatomic) CGFloat miterLimit
```

Discussion
The miter limit helps you avoid spikes in paths that use the `kCGLineJoinMiter` join style. If the ratio of the miter length—that is, the diagonal length of the miter join—to the line thickness exceeds the miter limit, the joint is converted to a bevel join. The default miter limit is 10, which results in the conversion of miters whose angle at the joint is less than 11 degrees.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

usesEvenOddFillRule

A Boolean indicating whether the even-odd winding rule is in use for drawing paths.

```
@property(nonatomic) BOOL usesEvenOddFillRule
```

Discussion
If `YES`, the path is filled using the even-odd rule. If `NO`, it is filled using the non-zero rule. Both rules are algorithms to determine which areas of a path to fill with the current fill color. A ray is drawn from a point inside a given region to a point anywhere outside the path's bounds. The total number of crossed path lines (including implicit path lines) and the direction of each path line are then interpreted as follows:

- For the even-odd rule, if the total number of path crossings is odd, the point is considered to be inside the path and the corresponding region is filled. If the number of crossings is even, the point is considered to be outside the path and the region is not filled.

- For the non-zero rule, the crossing of a left-to-right path counts as +1 and the crossing of a right-to-left path counts as -1. If the sum of the crossings is nonzero, the point is considered to be inside the path and the corresponding region is filled. If the sum is 0, the point is outside the path and the region is not filled.

The default value of this property is `NO`. For more information about winding rules and how they are applied to subpaths, see *Quartz 2D Programming Guide*.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

Class Methods

bezierPath

Creates and returns a new `UIBezierPath` object.

```
+ (UIBezierPath *)bezierPath
```

Return Value
A new empty path object.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

bezierPathWithArcCenter:radius:startAngle:endAngle:clockwise:

Creates and returns a new `UIBezierPath` object initialized with an arc of a circle.

```
+ (UIBezierPath *)bezierPathWithArcCenter:(CGPoint)center radius:(CGFloat)radius
    startAngle:(CGFloat)startAngle endAngle:(CGFloat)endAngle
    clockwise:(BOOL)clockwise
```

Parameters

`center`
Specifies the center point of the circle (in the current coordinate system) used to define the arc.

`radius`
Specifies the radius of the circle used to define the arc.

`startAngle`
Specifies the starting angle of the arc (measured in radians).

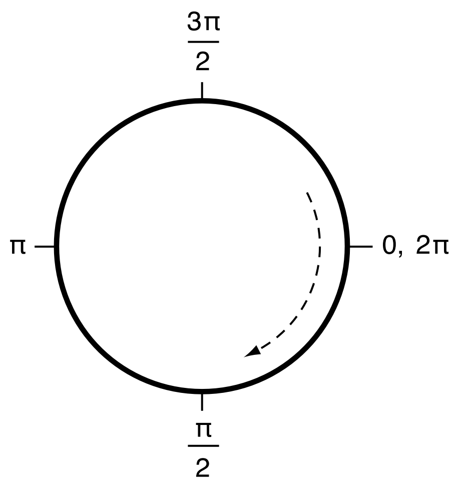
`endAngle`
Specifies the end angle of the arc (measured in radians).

`clockwise`
: The direction in which to draw the arc.

Return Value
A new path object with the specified arc.

Discussion
This method creates an open subpath. The created arc lies on the perimeter of the specified circle. When drawn in the default coordinate system, the start and end angles are based on the unit circle shown in Figure 1. For example, specifying a start angle of `0` radians, an end angle of `π` radians, and setting the `clockwise` parameter to `YES` draws the bottom half of the circle. However, specifying the same start and end angles but setting the *`clockwise`* parameter set to `NO` draws the top half of the circle.

Figure 1 Angles in the default coordinate system



After calling this method, the current point is set to the point on the arc at the end angle of the circle.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

bezierPathWithCGPath:

Creates and returns a new `UIBezierPath` object initialized with the contents of a Core Graphics path.

```
+ (UIBezierPath *)bezierPathWithCGPath:(CGPathRef)CGPath
```

Parameters

`CGPath`
: The Core Graphics path from which to obtain the initial path information. If this parameter is `nil`, the method raises an exception.

Return Value
A new path object with the specified path information.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

bezierPathWithOvalInRect:

Creates and returns a new `UIBezierPath` object initialized with an oval path inscribed in the specified rectangle

```
+ (UIBezierPath *)bezierPathWithOvalInRect:(CGRect)rect
```

Parameters

`rect`
The rectangle in which to inscribe an oval.

Return Value
A new path object with the oval path.

Discussion
This method creates a closed subpath that approximates the oval using a sequence of Bézier curves. The path is created in a clockwise direction (relative to the default coordinate system). If the *`rect`* parameter specifies a square, the inscribed path is a circle.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

bezierPathWithRect:

Creates and returns a new `UIBezierPath` object initialized with a rectangular path.

```
+ (UIBezierPath *)bezierPathWithRect:(CGRect)rect
```

Parameters

`rect`
The rectangle describing the path to create.

Return Value
A new path object with the rectangular path.

Discussion
This method creates a closed subpath by starting at the origin of *`rect`* and adding line segments in a clockwise direction (relative to the default coordinate system).

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

bezierPathWithRoundedRect:byRoundingCorners:cornerRadii:

Creates and returns a new `UIBezierPath` object initialized with a rounded rectangular path.

```
+ (UIBezierPath *)bezierPathWithRoundedRect:(CGRect)rect
    byRoundingCorners:(UIRectCorner)corners cornerRadii:(CGSize)cornerRadii
```

Parameters

rect

The rectangle that defines the basic shape of the path.

corners

A bitmask value that identifies the corners that you want rounded. You can use this parameter to round only a subset of the corners of the rectangle.

cornerRadii

The radius of each corner oval. Values larger than half the rectangle's width or height are clamped appropriately to half the width or height.

Return Value

A new path object with the rounded rectangular path.

Discussion

This method creates a closed subpath, proceeding in a clockwise direction (relative to the default coordinate system) as it creates the necessary line and curve segments.

Availability

Available in iOS 3.2 and later.

Declared In

`UIBezierPath.h`

bezierPathWithRoundedRect:cornerRadius:

Creates and returns a new `UIBezierPath` object initialized with a rounded rectangular path.

```
+ (UIBezierPath *)bezierPathWithRoundedRect:(CGRect)rect
    cornerRadius:(CGFloat)cornerRadius
```

Parameters

rect

The rectangle that defines the basic shape of the path

cornerRadius

The radius of each corner oval. A value of `0` results in a rectangle without rounded corners. Values larger than half the rectangle's width or height are clamped appropriately to half the width or height.

Return Value

A new path object with the rounded rectangular path.

Discussion

This method creates a closed subpath, proceeding in a clockwise direction (relative to the default coordinate system) as it creates the necessary line and curve segments.

Availability

Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

Instance Methods

addArcWithCenter:radius:startAngle:endAngle:clockwise:

Appends an arc to the receiver's path.

```
- (void)addArcWithCenter:(CGPoint)center radius:(CGFloat)radius
    startAngle:(CGFloat)startAngle endAngle:(CGFloat)endAngle
    clockwise:(BOOL)clockwise
```

Parameters

`center`
Specifies the center point of the circle (in the current coordinate system) used to define the arc.

`radius`
Specifies the radius of the circle used to define the arc.

`startAngle`
Specifies the starting angle of the arc (measured in radians).

`endAngle`
Specifies the end angle of the arc (measured in radians).

`clockwise`
The direction in which to draw the arc.

Discussion
This method adds the specified arc beginning at the current point. The created arc lies on the perimeter of the specified circle. When drawn in the default coordinate system, the start and end angles are based on the unit circle shown in Figure 1 (page 15). For example, specifying a start angle of `0` radians, an end angle of `π` radians, and setting the `clockwise` parameter to `YES` draws the bottom half of the circle. However, specifying the same start and end angles but setting the *`clockwise`* parameter set to `NO` draws the top half of the circle.

After calling this method, the current point is set to the point on the arc at the end angle of the circle.

Availability
Available in iOS 4.0 and later.

Declared In
`UIBezierPath.h`

addClip

Intersects the area enclosed by the receiver's path with the clipping path of the current graphics context and makes the resulting shape the current clipping path.

```
- (void)addClip
```

Discussion
This method modifies the visible drawing area of the current graphics context. After calling it, subsequent drawing operations result in rendered content only if they occur within the fill area of the specified path.

Important: If you need to remove the clipping region to perform subsequent drawing operations, you must save the current graphics state (using the `CGContextSaveGState` function) before calling this method. When you no longer need the clipping region, you can then restore the previous drawing properties and clipping region using the `CGContextRestoreGState` function.

The `usesEvenOddFillRule` (page 13) property is used to determine whether the even-odd or non-zero rule is used to determine the area enclosed by the path.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

addCurveToPoint:controlPoint1:controlPoint2:

Appends a cubic Bézier curve to the receiver's path.

```
- (void)addCurveToPoint:(CGPoint)endPoint controlPoint1:(CGPoint)controlPoint1
    controlPoint2:(CGPoint)controlPoint2
```

Parameters

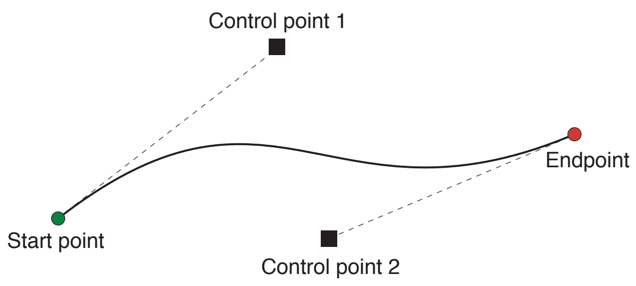
endPoint
The end point of the curve.

controlPoint1
The first control point to use when computing the curve.

controlPoint2
The second control point to use when computing the curve.

Discussion
This method appends a cubic Bézier curve from the current point to the end point specified by the *endPoint* parameter. The two control points define the curvature of the segment. Figure 2 shows an approximation of a cubic Bézier curve given a set of initial points. The exact curvature of the segment involves a complex mathematical relationship between all of the points and is well documented online.

Figure 2 A cubic Bézier curve



You must set the path's current point (using the `moveToPoint:` (page 25) method or through the previous creation of a line or curve segment) before you call this method. If the path is empty, this method does nothing. After adding the curve segment, this method updates the current point to the value in *`point`*.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

addLineToPoint:

Appends a straight line to the receiver's path.

```
- (void)addLineToPoint:(CGPoint)point
```

Parameters

`point`

The destination point of the line segment, specified in the current coordinate system.

Discussion
This method creates a straight line segment starting at the current point and ending at the point specified by the *`point`* parameter. After adding the line segment, this method updates the current point to the value in *`point`*.

You must set the path's current point (using the `moveToPoint:` (page 25) method or through the previous creation of a line or curve segment) before you call this method. If the path is empty, this method does nothing.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

addQuadCurveToPoint:controlPoint:

Appends a quadratic Bézier curve to the receiver's path.

```
- (void)addQuadCurveToPoint:(CGPoint)endPoint controlPoint:(CGPoint)controlPoint
```

Parameters

endPoint

The end point of the curve.

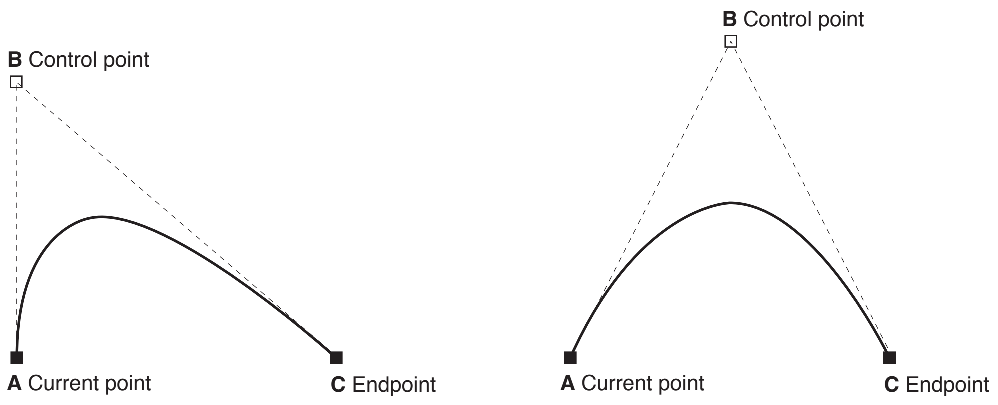
controlPoint

The control point of the curve.

Discussion

This method appends a quadratic Bézier curve from the current point to the end point specified by the *endPoint* parameter. The relationships between the current point, control point, and end point are what defines the actual curve. Figure 3 shows some examples of quadratic curves and the approximate curve shape based on some sample points. The exact curvature of the segment involves a complex mathematical relationship between the points and is well documented online.

Figure 3 Quadratic curve examples



You must set the path's current point (using the `moveToPoint:` (page 25) method or through the previous creation of a line or curve segment) before you call this method. If the path is empty, this method does nothing. After adding the curve segment, this method updates the current point to the value in *point*.

Availability

Available in iOS 3.2 and later.

Declared In

`UIBezierPath.h`

appendPath:

Appends the contents of the specified path object to the receiver's path.

```
- (void)appendPath:(UIBezierPath *)bezierPath
```

Parameters

bezierPath

The path to add to the receiver.

Discussion

This method adds the commands used to create the path in *bezierPath* to the end of the receiver's path. This method does not explicitly try to connect the subpaths in the two objects, although the operations in *bezierPath* might still cause that effect.

Availability

Available in iOS 3.2 and later.

Declared In

`UIBezierPath.h`

applyTransform:

Transforms all points in the path using the specified affine transform matrix.

```
- (void)applyTransform:(CGAffineTransform)transform
```

Parameters

transform

The transform matrix to apply to the path.

Discussion

This method applies the specified transform to the path's points immediately. The modifications made to the path object are permanent. If you do not want to permanently modify a path object, you should consider applying the transform to a copy.

Availability

Available in iOS 3.2 and later.

Declared In

`UIBezierPath.h`

closePath

Closes the most recently added subpath.

```
- (void)closePath
```

Discussion

This method closes the current subpath by creating a line segment between the first and last points in the subpath. This method subsequently updates the current point to the end of the newly created line segment, which is also the first point in the now closed subpath.

Availability

Available in iOS 3.2 and later.

Declared In

`UIBezierPath.h`

containsPoint:

Returns a Boolean value indicating whether the area enclosed by the receiver contains the specified point.

```
- (BOOL)containsPoint:(CGPoint)point
```

Parameters

point

The point to test against the path, specified in the path object's coordinate system.

Return Value

`YES` if the point is considered to be within the path's enclosed area or `NO` if it is not.

Discussion

The receiver contains the specified point if that point is in a portion of a closed subpath that would normally be painted during a fill operation. This method uses the value of the `usesEvenOddFillRule` (page 13) property to determine which parts of the subpath would be filled.

A point is not considered to be enclosed by the path if it is inside an open subpath, regardless of whether that area would be painted during a fill operation. Therefore, to determine mouse hits on open paths, you must create a copy of the path object and explicitly close any subpaths (using the `closePath` (page 22) method) before calling this method.

Availability

Available in iOS 3.2 and later.

Declared In

`UIBezierPath.h`

fill

Paints the region enclosed by the receiver's path using the current drawing properties.

```
- (void)fill
```

Discussion

This method fills the path using the current fill color and drawing properties. If the path contains any open subpaths, this method implicitly closes them before painting the fill region.

The painted region includes the pixels right up to, but not including, the path line itself. For paths with large line widths, this can result in overlap between the fill region and the stroked path (which is itself centered on the path line).

This method automatically saves the current graphics state prior to drawing and restores that state when it is done, so you do not have to save the graphics state yourself.

Availability

Available in iOS 3.2 and later.

Declared In

`UIBezierPath.h`

fillWithBlendMode:alpha:

Paints the region enclosed by the receiver's path using the specified blend mode and transparency values.

```
- (void)fillWithBlendMode:(CGBlendMode)blendMode alpha:(CGFloat)alpha
```

Parameters

blendMode

The blend mode determines how the filled path is composited with any existing rendered content.

alpha

The amount of transparency to apply to the filled path. Values can range between `0.0` (transparent) and `1.0` (opaque). Values outside this range are clamped to `0.0` or `1.0`.

Discussion

This method fills the path using the current fill color and drawing properties (plus the specified blend mode and transparency value). If the path contains any open subpaths, this method implicitly closes them before painting the fill region.

The painted region includes the pixels right up to, but not including, the path line itself. For paths with large line widths, this can result in overlap between the fill region and the stroked path (which is itself centered on the path line).

This method automatically saves the current graphics state prior to drawing and restores that state when it is done, so you do not have to save the graphics state yourself.

Availability

Available in iOS 3.2 and later.

Declared In

`UIBezierPath.h`

getLineDash:count:phase:

Retrieves the line-stroking pattern for the path.

```
- (void)getLineDash:(CGFloat *)pattern count:(NSInteger *)count phase:(CGFloat
    *)phase
```

Parameters

pattern

On input, a C-style array of floating point values, or `nil` if you do not want the pattern values. On output, this array contains the lengths (measured in points) of the line segments and gaps in the pattern. The values in the array alternate, starting with the first line segment length, followed by the first gap length, followed by the second line segment length, and so on.

count

On input, a pointer to an integer or `nil` if you do not want the number of pattern entries. On output, the number of entries written to *pattern*.

phase

On input, a pointer to a floating point value or `nil` if you do not want the phase. On output, this value contains the offset at which to start drawing the pattern, measured in points along the dashed-line pattern. For example, a phase of 6 in the pattern 5-2-3-2 would cause drawing to begin in the middle of the first gap.

Discussion
The array in the `pattern` parameter must be large enough to hold all of the returned values in the pattern. If you are not sure how many values there might be, you can call this method twice. The first time you call it, do not pass a value for `pattern` but use the returned value in the `count` parameter to allocate an array of floating-point numbers that you can then pass in the second time.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

moveToPoint:

Moves the receiver's current point to the specified location.

```
- (void)moveToPoint:(CGPoint)point
```

Parameters

`point`
A point in the current coordinate system.

Discussion
This method implicitly ends the current subpath (if any) and sets the current point to the value in the `point` parameter. When ending the previous subpath, this method does not actually close the subpath. Therefore, the first and last points of the previous subpath are not connected to each other.

For many path operations, you must call this method before issuing any commands that cause a line or curve segment to be drawn.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

removeAllPoints

Removes all points from the receiver, effectively deleting all subpaths.

```
- (void)removeAllPoints
```

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

setLineDash:count:phase:

Sets the line-stroking pattern for the path.

```
- (void)setLineDash:(const CGFloat *)pattern count:(NSInteger)count
    phase:(CGFloat)phase
```

Parameters

pattern

A C-style array of floating point values that contains the lengths (measured in points) of the line segments and gaps in the pattern. The values in the array alternate, starting with the first line segment length, followed by the first gap length, followed by the second line segment length, and so on.

count

The number of values in *pattern*.

phase

The offset at which to start drawing the pattern, measured in points along the dashed-line pattern. For example, a phase value of `6` for the pattern 5-2-3-2 would cause drawing to begin in the middle of the first gap.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

stroke

Draws a line along the receiver's path using the current drawing properties.

```
- (void)stroke
```

Discussion
The drawn line is centered on the path with its sides parallel to the path segment. This method applies the current drawing properties to the rendered path.

This method automatically saves the current graphics state prior to drawing and restores that state when it is done, so you do not have to save the graphics state yourself.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

strokeWithBlendMode:alpha:

Draws a line along the receiver's path using the specified blend mode and transparency values.

```
- (void)strokeWithBlendMode:(CGBlendMode)blendMode alpha:(CGFloat)alpha
```

Parameters

blendMode

The blend mode determines how the stroked path is composited with any existing rendered content.

alpha

The amount of transparency to apply to the stroked path. Values can range between `0.0` (transparent) and `1.0` (opaque). Values outside this range are clamped to `0.0` or `1.0`.

Discussion
The drawn line is centered on the path with its sides parallel to the path segment. This method applies the current stroke color and drawing properties (plus the specified blend mode and transparency value) to the rendered path.

This method automatically saves the current graphics state prior to drawing and restores that state when it is done, so you do not have to save the graphics state yourself.

Availability
Available in iOS 3.2 and later.

Declared In
`UIBezierPath.h`

Constants

UIRectCorner

The corners of a rectangle.

```
enum {
    UIRectCornerTopLeft     = 1 << 0,
    UIRectCornerTopRight    = 1 << 1,
    UIRectCornerBottomLeft  = 1 << 2,
    UIRectCornerBottomRight = 1 << 3,
    UIRectCornerAllCorners  = ~0
};
typedef NSUInteger UIRectCorner;
```

Constants

`UIRectCornerTopLeft`

The top-left corner of the rectangle.

Available in iOS 3.2 and later.

Declared in `UIBezierPath.h`.

`UIRectCornerTopRight`

The top-right corner of the rectangle.

Available in iOS 3.2 and later.

Declared in `UIBezierPath.h`.

`UIRectCornerBottomLeft`

The bottom-left corner of the rectangle.

Available in iOS 3.2 and later.

Declared in `UIBezierPath.h`.

`UIRectCornerBottomRight`

The bottom-right corner of the rectangle.

Available in iOS 3.2 and later.

Declared in `UIBezierPath.h`.

`UIRectCornerAllCorners`
    All corners of the rectangle.

    Available in iOS 3.2 and later.

    Declared in `UIBezierPath.h`.

Discussion

The specified constants reflect the corners of a rectangle that has not been modified by an affine transform and is drawn in the default coordinate system (where the origin is in the upper-left corner and positive values extend down and to the right).

Document Revision History

This table describes the changes to *UIBezierPath Class Reference*.

Date	Notes
2010-05-20	Added symbols introduced in iOS 4.0.
2010-03-12	New document descriing a vector-based path consisting of line and curve segments.