Quartz Core Framework Reference

Developer

Contents

Introduction

Framework	/System/Library/Frameworks/QuartzCore.framework
Header file directories	/System/Library/Frameworks/QuartzCore.framework/Headers
Declared in	CAAnimation.h CABase.h CADisplayLink.h CALayer.h CAMediaTiming.h CAMediaTimingFunction.h CAScrollLayer.h CATextLayer.h CATiledLayer.h CATransaction.h CATransform3D.h CIColor.h CIContext.h CIFilter.h CIImage.h CIVector.h UIImage.h
Companion guides	Core Image Programming Guide Image Unit Tutorial Core Image Kernel Language Reference Core Image Filter Reference Core Video Programming Guide

This collection of documents provides the API reference for the Quartz Core framework, which supports image processing and video image manipulation.

Classes

CAAnimation Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying CAAction CAMediaTiming NSObject (NSObject)
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in iOS 2.0 and later.
Declared in	CAAnimation.h
Companion guides	Core Animation Programming Guide Core Animation Cookbook
Related sample code	MoveMe

Overview

`CAAnimation` is an abstract animation class. It provides the basic support for the `CAMediaTiming` and `CAAction` protocols.

Tasks

Archiving Properties

– `shouldArchiveValueForKey:` (page 12)

Specifies whether the value of the property for a given key is archived.

Providing Default Values for Properties

+ `defaultValueForKey:` (page 11)
    Specifies the default value of the property with the specified key.

Creating an Animation

+ `animation` (page 11)
    Creates and returns a new `CAAnimation` instance.

Animation Attributes

`removedOnCompletion` (page 10) *property*
    Determines if the animation is removed from the target layer's animations upon completion.

`timingFunction` (page 10) *property*
    An optional timing function defining the pacing of the animation.

Getting and Setting the Delegate

`delegate` (page 9) *property*
    Specifies the receiver's delegate object.

Animation Progress

– `animationDidStart:` (page 13) *delegate method*
    Called when the animation begins its active duration.

– `animationDidStop:finished:` (page 13) *delegate method*
    Called when the animation completes its active duration or is removed from the object it is attached to.

Properties

delegate

Specifies the receiver's delegate object.

```
@property(retain) id delegate
```

Discussion

Defaults to `nil`.

> **Important:** The `delegate` object is retained by the receiver. This is a rare exception to the memory management rules described in *Advanced Memory Management Programming Guide*.
>
> An instance of `CAAnimation` should not be set as a delegate of itself. Doing so (outside of a garbage-collected environment) will cause retain cycles.

Availability

Available in iOS 2.0 and later.

Related Sample Code
MoveMe

Declared in

`CAAnimation.h`

removedOnCompletion

Determines if the animation is removed from the target layer's animations upon completion.

```
@property(getter=isRemovedOnCompletion) BOOL removedOnCompletion
```

Discussion

When `YES`, the animation is removed from the target layer's animations once its active duration has passed. Defaults to `YES`.

Availability

Available in iOS 2.0 and later.

Related Sample Code
MoveMe

Declared in

`CAAnimation.h`

timingFunction

An optional timing function defining the pacing of the animation.

```
@property(retain) CAMediaTimingFunction *timingFunction
```

Discussion
Defaults to `nil`, indicating linear pacing.

Availability
Available in iOS 2.0 and later.

Related Sample Code
MoveMe

Declared in
`CAAnimation.h`

Class Methods

animation

Creates and returns a new `CAAnimation` instance.

```
+ (id)animation
```

Return Value
An `CAAnimation` object whose input values are initialized.

Availability
Available in iOS 2.0 and later.

Related Sample Code
MoveMe

Declared in
`CAAnimation.h`

defaultValueForKey:

Specifies the default value of the property with the specified key.

```
+ (id)defaultValueForKey:(NSString *)key
```

Parameters
`key`

The name of one of the receiver's properties.

Return Value
The default value for the named property. Returns `nil` if no default value has been set.

Discussion
If this method returns `nil` a suitable "zero" default value for the property is provided, based on the declared type of the `key`. For example, if `key` is a `CGSize` object, a size of (0.0,0.0) is returned. For a `CGRect` an empty rectangle is returned. For `CGAffineTransform` and `CATransform3D`, the appropriate identity matrix is returned.

Special Considerations
If `key` is not a known for property of the class, the result of the method is undefined.

Availability
Available in iOS 2.0 and later.

Declared in
`CAAnimation.h`

Instance Methods

shouldArchiveValueForKey:

Specifies whether the value of the property for a given key is archived.

```
- (BOOL)shouldArchiveValueForKey:(NSString *)key
```

Parameters
`key`
    The name of one of the receiver's properties.

Return Value
`YES` if the specified property should be archived, otherwise `NO`.

Discussion
Called by the object's implementation of `encodeWithCoder:`. The object must implement keyed archiving.

The default implementation returns `YES`.

Availability
Available in iOS 4.0 and later.

Declared in
`CAAnimation.h`

Delegate Methods

animationDidStart:

Called when the animation begins its active duration.

```
- (void)animationDidStart:(CAAnimation *)theAnimation
```

Parameters

`theAnimation`
    The `CAAnimation` instance that started animating.

Availability
Available in iOS 2.0 and later.

Declared in
`CAAnimation.h`

animationDidStop:finished:

Called when the animation completes its active duration or is removed from the object it is attached to.

```
- (void)animationDidStop:(CAAnimation *)theAnimation
finished:(BOOL)flag
```

Parameters

`theAnimation`
    The `CAAnimation` instance that stopped animating.

`flag`
    If `YES`, the animation reached the end of its active duration without being removed.

Availability
Available in iOS 2.0 and later.

Declared in
`CAAnimation.h`

CAAnimationGroup Class Reference

Inherits from	CAAnimation : NSObject
Conforms to	NSCoding (CAAnimation) NSCopying (CAAnimation) CAAction (CAAnimation) CAMediaTiming (CAAnimation) NSObject (NSObject)
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in iOS 2.0 and later.
Declared in	CAAnimation.h
Companion guides	Core Animation Programming Guide Core Animation Cookbook
Related sample code	MoveMe

Overview

`CAAnimationGroup` allows multiple animations to be grouped and run concurrently. The grouped animations run in the time space specified by the `CAAnimationGroup` instance.

The duration of the grouped animations are not scaled to the duration of their CAAnimationGroup. Instead, the animations are clipped to the duration of the animation group. For example, a 10 second animation grouped within an animation group with a duration of 5 seconds will only display the first 5 seconds of the animation.

Important: The `delegate` and `removedOnCompletion` properties of animations in the `animations` (page 15) array are currently ignored. The `CAAnimationGroup` delegate does receive these messages.

Note: The `delegate` and `removedOnCompletion` properties of animations in the `animations` (page 15) property are currently ignored.

Tasks

Grouped Animations

`animations` (page 15) *property*
An array of `CAAnimation` objects to be evaluated in the time space of the receiver.

Properties

animations

An array of `CAAnimation` objects to be evaluated in the time space of the receiver.

```
@property(copy) NSArray *animations
```

Discussion

The animations run concurrently in the receiver's time space.

Availability

Available in iOS 2.0 and later.

Related Sample Code
MoveMe

Declared in

`CAAnimation.h`

CABasicAnimation Class Reference

Inherits from	CAPropertyAnimation : CAAnimation : NSObject
Conforms to	NSCoding (CAAnimation) NSCopying (CAAnimation) CAAction (CAAnimation) CAMediaTiming (CAAnimation) NSObject (NSObject)
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in iOS 2.0 and later.
Declared in	CAAnimation.h
Companion guides	Core Animation Programming Guide Core Animation Cookbook
Related sample code	iAdInterstitialSuite MoveMe

Overview

`CABasicAnimation` provides basic, single-keyframe animation capabilities for a layer property. You create an instance of `CABasicAnimation` using the inherited `animationWithKeyPath:` (page 111) method, specifying the key path of the property to be animated in the render tree.

Setting Interpolation Values

The `fromValue` (page 18), `byValue` (page 17) and `toValue` (page 18) properties define the values being interpolated between. All are optional, and no more than two should be non-`nil`. The object type should match the type of the property being animated.

The interpolation values are used as follows:

- Both `fromValue` (page 18) and `toValue` (page 18) are non-`nil`. Interpolates between `fromValue` (page 18) and `toValue` (page 18).
- `fromValue` (page 18) and `byValue` (page 17) are non-`nil`. Interpolates between `fromValue` (page 18) and (`fromValue` (page 18) + `byValue` (page 17)).
- `byValue` (page 17) and `toValue` (page 18) are non-`nil`. Interpolates between (`toValue` (page 18) - `byValue` (page 17)) and `toValue` (page 18).
- `fromValue` (page 18) is non-`nil`. Interpolates between `fromValue` (page 18) and the current presentation value of the property.
- `toValue` (page 18) is non-`nil`. Interpolates between the current value of `keyPath` in the target layer's presentation layer and `toValue` (page 18).
- `byValue` (page 17) is non-`nil`. Interpolates between the current value of `keyPath` in the target layer's presentation layer and that value plus `byValue` (page 17).
- All properties are `nil`. Interpolates between the previous value of `keyPath` in the target layer's presentation layer and the current value of `keyPath` in the target layer's presentation layer.

Tasks

Interpolation Values

`fromValue` (page 18) *property*
Defines the value the receiver uses to start interpolation.

`toValue` (page 18) *property*
Defines the value the receiver uses to end interpolation.

`byValue` (page 17) *property*
Defines the value the receiver uses to perform relative interpolation.

Properties

byValue

Defines the value the receiver uses to perform relative interpolation.

```
@property(retain) id byValue
```

Discussion

See "Setting Interpolation Values" (page 16) for details on how `byValue` interacts with the other interpolation values.

Availability

Available in iOS 2.0 and later.

Declared in

`CAAnimation.h`

fromValue

Defines the value the receiver uses to start interpolation.

```
@property(retain) id fromValue
```

Discussion

See "Setting Interpolation Values" (page 16) for details on how `fromValue` interacts with the other interpolation values.

Availability

Available in iOS 2.0 and later.

Declared in

`CAAnimation.h`

toValue

Defines the value the receiver uses to end interpolation.

```
@property(retain) id toValue
```

Discussion

See "Setting Interpolation Values" (page 16) for details on how `toValue` interacts with the other interpolation values.

Availability

Available in iOS 2.0 and later.

Related Sample Code

MoveMe

Declared in
`CAAnimation.h`

CADisplayLink Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in iOS 3.1 and later.
Declared in	CADisplayLink.h
Related sample code	GLAirplay pARk Real-time Video Processing Using AVPlayerItemVideoOutput

Overview

A `CADisplayLink` object is a timer object that allows your application to synchronize its drawing to the refresh rate of the display.

Your application creates a new display link, providing a target object and a selector to be called when the screen is updated. Next, your application adds the display link to a run loop.

Once the display link is associated with a run loop, the selector on the target is called when the screen's contents need to be updated. The target can read the display link's `timestamp` (page 23) property to retrieve the time that the previous frame was displayed. For example, an application that displays movies might use the timestamp to calculate which video frame will be displayed next. An application that performs its own animations might use the timestamp to determine where and how displayed objects appear in the upcoming frame. The `duration` (page 22) property provides the amount of time between frames. You can use this value in your application to calculate the frame rate of the display, the approximate time that the next frame will be displayed, and to adjust the drawing behavior so that the next frame is prepared in time to be displayed.

Your application can disable notifications by setting the `paused` (page 23) property to `YES`. Also, if your application cannot provide frames in the time provided, you may want to choose a slower frame rate. An application with a slower but consistent frame rate appears smoother to the user than an application that skips frames. You can increase the time between frames (and decrease the apparent frame rate) by changing the `frameInterval` (page 22) property.

When your application finishes with a display link, it should call `invalidate` (page 25) to remove it from all run loops and to disassociate it from the target.

`CADisplayLink` should not be subclassed.

Tasks

Creating Instances

`+ displayLinkWithTarget:selector:` (page 23)
Returns a new display link.

Scheduling the Display Link to Send Notifications

`– addToRunLoop:forMode:` (page 24)
Registers the display link with a run loop.

`– removeFromRunLoop:forMode:` (page 25)
Removes the display link from the run loop for the given mode.

`– invalidate` (page 25)
Removes the display link from all run loop modes.

Configuring the Display Link

`duration` (page 22) *property*
The time interval between screen refresh updates. (read-only)

`frameInterval` (page 22) *property*
The number of frames that must pass before the display link notifies the target again.

`paused` (page 23) *property*
A Boolean value that states whether the display link’s notifications to the target are suspended.

`timestamp` (page 23) *property*
The time value associated with the last frame that was displayed. (read-only)

Properties

duration

The time interval between screen refresh updates. (read-only)

```
@property(readonly, nonatomic) CFTimeInterval duration
```

Discussion
The value for duration is undefined before the target's selector has been called at least once. Your application can calculate the amount of time it has to render each frame by multiplying `duration` by `frameInterval` (page 22).

Availability
Available in iOS 3.1 and later.

Declared in
`CADisplayLink.h`

frameInterval

The number of frames that must pass before the display link notifies the target again.

```
@property(nonatomic) NSInteger frameInterval
```

Discussion
The default value is `1`, which results in your application being notified at the refresh rate of the display. If the value is set to a value larger than `1`, the display link notifies your application at a fraction of the native refresh rate. For example, setting the interval to `2` causes the display link to fire every other frame, providing half the frame rate.

Setting this value to less than `1` results in undefined behavior and is a programmer error.

Availability
Available in iOS 3.1 and later.

Declared in
`CADisplayLink.h`

paused

A Boolean value that states whether the display link's notifications to the target are suspended.

```
@property(getter=isPaused, nonatomic) BOOL paused
```

Discussion
The default value is `NO`. If `YES`, the display link does not send notifications to the target.

Availability
Available in iOS 3.1 and later.

Declared in
`CADisplayLink.h`

timestamp

The time value associated with the last frame that was displayed. (read-only)

```
@property(readonly, nonatomic) CFTimeInterval timestamp
```

Discussion
The target should use the value of this property to calculate what should be displayed in the next frame.

Availability
Available in iOS 3.1 and later.

Related Sample Code
Real-time Video Processing Using AVPlayerItemVideoOutput

Declared in
`CADisplayLink.h`

Class Methods

displayLinkWithTarget:selector:

Returns a new display link.

```
+ (CADisplayLink *)displayLinkWithTarget:(id)target selector:(SEL)sel
```

Parameters

`target`

An object to be notified when the screen should be updated.

`sel`

The method to call on the target.

Return Value

A newly constructed display link.

Discussion

The selector to be called on the target must be a method with the following signature:

```
- (void) selector:(CADisplayLink *)sender;
```

where *sender* is the display link returned by this method.

The newly constructed display link retains the target.

Availability

Available in iOS 3.1 and later.

Related Sample Code
GLAirplay
pARk
Real-time Video Processing Using AVPlayerItemVideoOutput

Declared in

`CADisplayLink.h`

Instance Methods

addToRunLoop:forMode:

Registers the display link with a run loop.

`- (void)addToRunLoop:(NSRunLoop *)runloop forMode:(NSString *)mode`

Parameters

`runloop`

The run loop to associate with the display link.

`mode`

The mode in which to add the display link to the run loop. You may specify a custom mode or use one of the modes listed in *NSRunLoop Class Reference* .

Discussion

You can associate a display link with multiple input modes. While the run loop is executing in a mode you have specified, the display link notifies the target when new frames are required.

The run loop retains the display link. To remove the display link from all run loops, send an `invalidate` (page 25) message to the display link.

Availability

Available in iOS 3.1 and later.

See Also

– `removeFromRunLoop:forMode:` (page 25)

Declared in

`CADisplayLink.h`

invalidate

Removes the display link from all run loop modes.

```
- (void)invalidate
```

Discussion

Removing the display link from all run loop modes causes it to be released by the run loop. The display link also releases the target.

Availability

Available in iOS 3.1 and later.

Declared in

`CADisplayLink.h`

removeFromRunLoop:forMode:

Removes the display link from the run loop for the given mode.

```
- (void)removeFromRunLoop:(NSRunLoop *)runloop forMode:(NSString *)mode
```

Parameters

`runloop`

The run loop associated with the display link.

`mode`

The run loop mode in which the display link is running.

Discussion

The run loop releases the display link if it is no longer associated with any run modes.

Availability

Available in iOS 3.1 and later.

See Also

`– addToRunLoop:forMode:` (page 24)

Declared in

`CADisplayLink.h`

CAEAGLLayer Class Reference

Inherits from	CALayer : NSObject
Conforms to	EAGLDrawable NSCoding (CALayer) CAMediaTiming (CALayer) NSObject (NSObject)
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in iOS 2.0 and later.
Declared in	CAEAGLLayer.h
Related sample code	GLAirplay GLGravity Real-time Video Processing Using AVPlayerItemVideoOutput RosyWriter SpeakHere

Overview

The `CAEAGLLayer` class supports drawing OpenGL content in iPhone applications. If you plan to use OpenGL for your rendering, use this class as the backing layer for your views by returning it from your view's `layerClass` class method. The returned `CAEAGLLayer` object is a wrapper for a Core Animation surface that is fully compatible with OpenGL ES function calls.

Prior to designating the layer's associated view as the render target for a graphics context, you can change the rendering attributes you want using the `drawableProperties` property. This property lets you configure the color format for the rendering surface and whether the surface retains its contents.

Because an OpenGL ES rendering surface is presented to the user using Core Animation, any effects and animations you apply to the layer affect the 3D content you render. However, for best performance, do the following:

- Set the layer's opaque attribute to `TRUE`.

- Set the layer bounds to match the dimensions of the display.
- Make sure the layer is not transformed.
- Avoid drawing other layers on top of the `CAEAGLLayer` object. If you must draw other, non OpenGL content, you might find the performance cost acceptable if you place transparent 2D content on top of the GL content and also make sure that the OpenGL content is opaque and not transformed.
- When drawing landscape content on a portrait display, you should rotate the content yourself rather than using the `CAEAGLLayer` transform to rotate it.

Tasks

Accessing the Layer Properties

`drawableProperties` (page 28) *property*
The properties of the native windowing surface.

Properties

drawableProperties

The properties of the native windowing surface.

```
@property(copy) NSDictionary *drawableProperties
```

Discussion

You can use this property to change the underlying color format for the windowing surface and whether or not the surface retains its contents. For a list of keys (and corresponding values) you can include in this dictionary (along with their default values), see the *EAGLDrawable Protocol Reference* .

CAKeyframeAnimation Class Reference

Inherits from	CAPropertyAnimation : CAAnimation : NSObject
Conforms to	NSCoding (CAAnimation) NSCopying (CAAnimation) CAAction (CAAnimation) CAMediaTiming (CAAnimation) NSObject (NSObject)
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in iOS 2.0 and later.
Declared in	CAAnimation.h
Companion guides	Core Animation Programming Guide Core Animation Cookbook
Related sample code	MoveMe

Overview

The `CAKeyframeAnimation` class provides keyframe animation capabilities for a layer object. You create an `CAKeyframeAnimation` object using the inherited `animationWithKeyPath:` (page 111) method, specifying the key path of the property that you want to animate on the layer. You can then specify the keyframe values to use to control the timing and animation behavior.

For most types of animations, you specify the keyframe values using the `values` (page 35) and `keyTimes` (page 32) properties. During the animation, Core Animation generates intermediate values by interpolating between the values you provide. When animating a value that is a coordinate point, such as the layer's position, you can specify a `path` (page 33) for that point to follow instead of individual values. The pacing of the animation is controlled by the timing information you provide.

Tasks

Providing Keyframe Values

`values` (page 35) *property*
An array of objects that specify the keyframe values to use for the animation.

`path` (page 33) *property*
The path for a point-based property to follow.

Keyframe Timing

`keyTimes` (page 32) *property*
An optional array of `NSNumber` objects that define the time at which to apply a given keyframe segment.

`timingFunctions` (page 34) *property*
An optional array of `CAMediaTimingFunction` objects that define the pacing for each keyframe segment.

`calculationMode` (page 31) *property*
Specifies how intermediate keyframe values are calculated by the receiver.

Rotation Mode Attribute

`rotationMode` (page 33) *property*
Determines whether objects animating along the path rotate to match the path tangent.

Cubic Mode Attributes

`tensionValues` (page 34) *property*
An array of `NSNumber` objects that define the tightness of the curve.

`continuityValues` (page 31) *property*
An array of `NSNumber` objects that define the sharpness of the timing curve's corners.

`biasValues` (page 31) *property*
An array of `NSNumber` objects that define the position of the curve relative to a control point.

Properties

biasValues

An array of `NSNumber` objects that define the position of the curve relative to a control point.

```
@property(copy) NSArray *biasValues
```

Discussion
This property is used only for the cubic calculation modes. Positive values move the curve before the control point while negative values move it after the control point. The first value defines the behavior of the tangent to the first control point, the second value controls the second point's tangents, and so on. If you do not specify a value for a given control point, the value `0` is used.

Availability
Available in iOS 4.0 and later.

Declared in
`CAAnimation.h`

calculationMode

Specifies how intermediate keyframe values are calculated by the receiver.

```
@property(copy) NSString *calculationMode
```

Discussion
The possible values are described in "`Value calculation modes`" (page 36). The default value of this property is `kCAAnimationLinear` (page 36).

Availability
Available in iOS 2.0 and later.

Declared in
`CAAnimation.h`

continuityValues

An array of `NSNumber` objects that define the sharpness of the timing curve's corners.

```
@property(copy) NSArray *continuityValues
```

Discussion

This property is used only for the cubic calculation modes. Positive values result in sharper corners while negative values create inverted corners. The first value defines the behavior of the tangent to the first control point, the second value controls the second point's tangents, and so on. If you do not specify a value for a given control point, the value `0` is used.

Availability

Available in iOS 4.0 and later.

Declared in

`CAAnimation.h`

keyTimes

An optional array of `NSNumber` objects that define the time at which to apply a given keyframe segment.

```
@property(copy) NSArray *keyTimes
```

Discussion

Each value in the array is a floating point number between `0.0` and `1.0` that defines the time point (specified as a fraction of the animation's total duration) at which to apply the corresponding keyframe value. Each successive value in the array must be greater than, or equal to, the previous value. Usually, the number of elements in the array should match the number of elements in the `values` (page 35) property or the number of control points in the `path` (page 33) property. If they do not, the timing of your animation might not be what you expect.

The appropriate values to include in the array are dependent on the `calculationMode` (page 31) property.

- If the calculationMode is set to `kCAAnimationLinear` or `kCAAnimationCubic`, the first value in the array must be `0.0` and the last value must be `1.0`. All intermediate values represent time points between the start and end times.
- If the calculationMode is set to `kCAAnimationDiscrete`, the first value in the array must be `0.0` and the last value must be `1.0`. The array should have one more entry than appears in the values array. For example, if there are two values, there should be three key times.
- If the calculationMode is set to `kCAAnimationPaced` or `kCAAnimationCubicPaced`, the values in this property are ignored.

If the values in this array are invalid or inappropriate for the current calculation mode, they are ignored.

Availability

Available in iOS 2.0 and later.

Declared in
`CAAnimation.h`

path

The path for a point-based property to follow.

```
@property CGPathRef path;
```

Discussion
For layer properties that contain a `CGPoint` data type, the path object you assign to this property defines the values for that property over the length of the animation. If you specify a value for this property, any data in the `values` (page 35) property is ignored.

Any timing values you specify for the animation are applied to the points used to create the path. Paths can contain points defining move-to, line-to, or curve-to segments. The end point of a line-to or curve-to segment defines the keyframe value. All other points between that end value and the previous value are then interpolated. Move-to segments do not define separate keyframe values.

How the animation proceeds along the path is dependent on the value in the `calculationMode` (page 31) property. To achieve a smooth, constant velocity animation along the path, set the `calculationMode` property to `kCAAnimationPaced` (page 37) or `kCAAnimationCubicPaced` (page 37). To create an animation where the location value jumps from keyframe point to keyframe point (without interpolation in between), use the `kCAAnimationDiscrete` (page 37) value. To animate along the path by interpolating values between points, use the `kCAAnimationLinear` (page 36) value.

Availability
Available in iOS 2.0 and later.

See Also
`@property rotationMode` (page 33)

Related Sample Code
MoveMe

Declared in
`CAAnimation.h`

rotationMode

Determines whether objects animating along the path rotate to match the path tangent.

```
@property(copy) NSString *rotationMode
```

Discussion

The possible values for this property are described in `"Rotation Mode Values"` (page 36). The default value of this property is `nil`, which indicates that objects should not rotate to follow the path.

The effect of setting this property to a non-`nil` value when no path object is supplied is undefined.

Availability

Available in iOS 2.0 and later.

See Also

`@property path` (page 33)

Declared in

`CAAnimation.h`

tensionValues

An array of `NSNumber` objects that define the tightness of the curve.

```
@property(copy) NSArray *tensionValues
```

Discussion

This property is used only for the cubic calculation modes. Positive values indicate a tighter curve while negative values indicate a rounder curve. The first value defines the behavior of the tangent to the first control point, the second value controls the second point's tangents, and so on. If you do not specify a value for a given control point, the value `0` is used.

Availability

Available in iOS 4.0 and later.

Declared in

`CAAnimation.h`

timingFunctions

An optional array of `CAMediaTimingFunction` objects that define the pacing for each keyframe segment.

```
@property(copy) NSArray *timingFunctions
```

Discussion
You can use this array to apply ease-in, ease-out, or custom timing curves to the points that lie between two keyframe values. If the number of keyframes in the values property is *n*, then this property should contain *n*−1 objects.

If you provide timing information in the `keyTimes` (page 32) property, the timing functions you specify using this property further modify the timing between those values. If you do not assign a value to the `keyTimes` property, the timing functions modify the default timing provided by the animation object.

If you also specify a timing function in the animation object's `timingFunction` (page 10) property, that function is applied first followed by the timing function for the specific keyframe segment.

For information on how to create a timing function, see *CAMediaTimingFunction Class Reference*.

Availability
Available in iOS 2.0 and later.

Declared in
`CAAnimation.h`

values

An array of objects that specify the keyframe values to use for the animation.

```
@property(copy) NSArray *values
```

Discussion
The keyframe values represent the values through which the animation must proceed. The time at which a given keyframe value is applied to the layer depends on the animation timing, which is controlled by the `calculationMode` (page 31), `keyTimes` (page 32), and `timingFunctions` (page 34) properties. Values between keyframes are created using interpolation, unless the calculation mode is set to `kCAAnimationDiscrete` (page 37).

Depending on the type of the property, you may need to wrap the values in this array with an `NSNumber` of `NSValue` object. For some Core Graphics data types, you may also need to cast them to `id` before adding them to the array.

The values in this property are used only if the value in the `path` (page 33) property is `nil`.

Availability
Available in iOS 2.0 and later.

Declared in
`CAAnimation.h`

Constants

Rotation Mode Values

These constants are used by the `rotationMode` (page 33) property.

```
NSString * const kCAAnimationRotateAuto
NSString * const kCAAnimationRotateAutoReverse
```

Constants

`kCAAnimationRotateAuto`

The objects travel on a tangent to the path.

Available in iOS 2.0 and later.

Declared in `CAAnimation.h`.

`kCAAnimationRotateAutoReverse`

The objects travel at a 180 degree tangent to the path.

Available in iOS 2.0 and later.

Declared in `CAAnimation.h`.

Value calculation modes

These constants are used by the `calculationMode` (page 31) property.

```
NSString * const kCAAnimationLinear;
NSString * const kCAAnimationDiscrete;
NSString * const kCAAnimationPaced;
NSString * const kCAAnimationCubic;
NSString * const kCAAnimationCubicPaced;
```

Constants

`kCAAnimationLinear`

Simple linear calculation between keyframe values.

Available in iOS 2.0 and later.

Declared in `CAAnimation.h`.

`kCAAnimationDiscrete`
: Each keyframe value is used in turn, no interpolated values are calculated.

  Available in iOS 2.0 and later.

  Declared in `CAAnimation.h`.

`kCAAnimationPaced`
: Keyframe values are interpolated to produce an even pace throughout the animation.

  Available in iOS 2.0 and later.

  Declared in `CAAnimation.h`.

`kCAAnimationCubic`
: Intermediate frames are computed using a Catmull-Rom spline that passes through the keyframes. You can adjust the shape of the spline by specifying an optional set of tension, continuity, and bias values, which modify the spline using the standard Kochanek-Bartels form.

  Available in iOS 4.0 and later.

  Declared in `CAAnimation.h`.

`kCAAnimationCubicPaced`
: Intermediate frames are computed using the cubic scheme but the `keyTimes` and `timingFunctions` properties of the animation are ignored. Instead, timing parameters are calculated implicitly to give the animation a constant velocity.

  Available in iOS 4.0 and later.

  Declared in `CAAnimation.h`.

CALayer Class Reference

Inherits from	NSObject
Conforms to	NSCoding CAMediaTiming NSObject (NSObject)
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in iOS 2.0 and later.
Declared in	CAConstraintLayoutManager.h CALayer.h CAScrollLayer.h
Companion guides	Core Animation Programming Guide Core Animation Cookbook
Related sample code	AccelerometerGraph AVSimpleEditoriOS MotionGraphs MTAudioProcessingTap Audio Processor oalTouch

Overview

The `CALayer` class manages image-based content and allows you to perform animations on that content. Layers are often used to provide the backing store for views but can also be used without a view to display content. A layer's main job is to manage the visual content that you provide but the layer itself has visual attributes that can be set, such as a background color, border, and shadow. In addition to managing visual content, the layer also maintains information about the geometry of its content (such as its position, size, and transform) that is used to present that content onscreen. Modifying the properties of the layer is how you initiate animations on the layer's content or geometry. A layer object encapsulates the duration and pacing of a layer and its animations by adopting the `CAMediaTiming` protocol, which defines the layer's timing information.

If the layer object was created by a view, the view typically assigns itself as the layer's delegate automatically, and you should not change that relationship. For layers you create yourself, you can assign a `delegate` (page 55) object and use that object to provide the contents of the layer dynamically and perform other tasks. A layer may also have a layout manager object (assigned to the `layoutManager` property) to manage the layout of subviews separately.

Tasks

Creating a Layer

`+ layer` (page 73)
Creates and returns an instance of the layer object.

`– init` (page 85)
Returns an initialized `CALayer` object.

`– initWithLayer:` (page 85)
Override to copy or initialize custom fields of the specified layer.

Accessing Related Layer Objects

`– presentationLayer` (page 90)
Returns a copy of the presentation layer object that represents the state of the layer as it currently appears onscreen.

`– modelLayer` (page 88)
Returns the model layer object associated with the receiver, if any.

Accessing the Delegate

`delegate` (page 55) *property*
The layer's delegate object.

Providing the Layer’s Content

`contents` (page 52) *property*
: An object that provides the contents of the layer. Animatable.

`contentsRect` (page 54) *property*
: The rectangle, in the unit coordinate space, that defines the portion of the layer’s contents that should be used. Animatable.

`contentsCenter` (page 53) *property*
: The rectangle that defines how the layer contents are scaled during a resizing operation. Animatable.

– `display` (page 83)
: Reloads the content of this layer.

– `drawInContext:` (page 84)
: Draws the layer’s content using the specified graphics context.

Modifying the Layer’s Appearance

`contentsGravity` (page 53) *property*
: A constant that specifies how the layer's contents are positioned or scaled within its bounds.

`opacity` (page 63) *property*
: The opacity of the receiver. Animatable.

`hidden` (page 59) *property*
: A Boolean indicating whether the layer is displayed. Animatable.

`masksToBounds` (page 61) *property*
: A Boolean indicating whether sublayers are clipped to the layer’s bounds. Animatable.

`mask` (page 60) *property*
: An optional layer whose alpha channel is used to mask the layer’s content.

`doubleSided` (page 56) *property*
: A Boolean indicating whether the layer displays its content when facing away from the viewer. Animatable.

`cornerRadius` (page 55) *property*
: The radius to use when drawing rounded corners for the layer’s background. Animatable.

`borderWidth` (page 50) *property*
: The width of the layer’s border. Animatable.

`borderColor` (page 49) *property*
: The color of the layer’s border. Animatable.

`backgroundColor` (page 48) *property*
: The background color of the receiver. Animatable.

`shadowOpacity` (page 66) *property*
: The opacity of the layer's shadow. Animatable.

`shadowRadius` (page 67) *property*
: The blur radius (in points) used to render the layer's shadow. Animatable.

`shadowOffset` (page 66) *property*
: The offset (in points) of the layer's shadow. Animatable.

`shadowColor` (page 65) *property*
: The color of the layer's shadow. Animatable.

`shadowPath` (page 67) *property*
: The shape of the layer's shadow. Animatable.

`style` (page 68) *property*
: An optional dictionary used to store property values that aren't explicitly defined by the layer.

Accessing the Layer's Filters

`filters` (page 57) *property*
: An array of Core Image filters to apply to the contents of the layer and its sublayers. Animatable.

`compositingFilter` (page 51) *property*
: A CoreImage filter used to composite the layer and the content behind it. Animatable.

`backgroundFilters` (page 49) *property*
: An array of Core Image filters to apply to the content immediately behind the layer. Animatable.

`minificationFilter` (page 61) *property*
: The filter used when reducing the size of the content.

`minificationFilterBias` (page 62) *property*
: The bias factor used by the minification filter to determine the levels of detail.

`magnificationFilter` (page 60) *property*
: The filter used when increasing the size of the content.

Configuring the Layer's Rendering Behavior

`opaque` (page 63) *property*
: A Boolean value indicating whether the layer contains completely opaque content.

`edgeAntialiasingMask` (page 57) *property*
: A bitmask defining how the edges of the receiver are rasterized.

– `contentsAreFlipped` (page 79)
: Returns a Boolean indicating whether the layer content is implicitly flipped when rendered.

`geometryFlipped` (page 59) *property*
: A Boolean that indicates whether the geometry of the layer and its sublayers is flipped vertically.

`drawsAsynchronously` (page 56) *property*
: A Boolean indicating whether drawing commands are deferred and processed asynchronously in a background thread.

`shouldRasterize` (page 68) *property*
: A Boolean that indicates whether the layer is rendered as a bitmap before compositing. Animatable

`rasterizationScale` (page 65) *property*
: The scale at which to rasterize content, relative to the coordinate space of the layer. Animatable

– `renderInContext:` (page 92)
: Renders the receiver and its sublayers into the specified context.

Modifying the Layer Geometry

`frame` (page 58) *property*
: The layer's frame rectangle.

`bounds` (page 50) *property*
: The layer's bounds rectangle. Animatable.

`position` (page 64) *property*
: The layer's position in its superlayer's coordinate space. Animatable.

`zPosition` (page 71) *property*
: The layer's position on the z axis. Animatable.

`anchorPointZ` (page 48) *property*
: The anchor point for the layer's position along the z axis. Animatable.

`anchorPoint` (page 47) *property*
: Defines the anchor point of the layer's bounds rectangle. Animatable.

`contentsScale` (page 54) *property*
: The scale factor applied to the layer.

Managing the Layer's Transform

`transform` (page 70) *property*

The transform applied to the layer's contents. Animatable.

`sublayerTransform` (page 70) *property*

Specifies the transform to apply to sublayers when rendering. Animatable.

– `affineTransform` (page 77)

Returns an affine version of the layer's transform.

– `setAffineTransform:` (page 94)

Sets the layer's transform to the specified affine transform.

Managing the Layer Hierarchy

`sublayers` (page 69) *property*

An array containing the layer's sublayers.

`superlayer` (page 70) *property*

The superlayer of the layer. (read-only)

– `addSublayer:` (page 76)

Appends the layer to the layer's list of sublayers.

– `removeFromSuperlayer` (page 91)

Detaches the layer from its parent layer.

– `insertSublayer:atIndex:` (page 86)

Inserts the specified layer into the receiver's list of sublayers at the specified index.

– `insertSublayer:below:` (page 87)

Inserts the specified sublayer below a different sublayer that already belongs to the receiver.

– `insertSublayer:above:` (page 86)

Inserts the specified sublayer above a different sublayer that already belongs to the receiver.

– `replaceSublayer:with:` (page 92)

Replaces the specified sublayer with a different layer object.

Updating Layer Display

– `setNeedsDisplay` (page 94)

Marks the layer's contents as needing to be updated.

– `setNeedsDisplayInRect:` (page 95)
Marks the region within the specified rectangle as needing to be updated.

`needsDisplayOnBoundsChange` (page 63) *property*
A Boolean indicating whether the layer contents must be updated when its bounds rectangle changes.

– `displayIfNeeded` (page 83)
Initiates the update process for a layer if it is currently marked as needing an update.

– `needsDisplay` (page 89)
Returns a Boolean indicating whether the layer has been marked as needing an update.

+ `needsDisplayForKey:` (page 74)
Returns a Boolean indicating whether changes to the specified key require the layer to be redisplayed.

Layer Animations

– `addAnimation:forKey:` (page 76)
Add the specified animation object to the layer's render tree.

– `animationForKey:` (page 77)
Returns the animation object with the specified identifier.

– `removeAllAnimations` (page 91)
Remove all animations attached to the layer.

– `removeAnimationForKey:` (page 91)
Remove the animation object with the specified key.

– `animationKeys` (page 78)
Returns an array of strings that identify the animations currently attached to the layer.

Managing Layer Resizing and Layout

– `setNeedsLayout` (page 95)
Invalidates the layer's layout and marks it as needing an update.

– `layoutSublayers` (page 88)
Tells the layer to update its layout.

– `layoutIfNeeded` (page 87)
Recalculate the receiver's layout, if required.

– `needsLayout` (page 89)
Returns a Boolean indicating whether the layer has been marked as needing a layout update.

– `preferredFrameSize` (page 90)
Returns the preferred size of the layer in the coordinate space of its superlayer.

Getting the Layer's Actions

– `actionForKey:` (page 74)
Returns the action object assigned to the specified key.

`actions` (page 46) *property*
A dictionary containing layer actions.

+ `defaultActionForKey:` (page 72)
Returns the default action for the current class.

Mapping Between Coordinate and Time Spaces

– `convertPoint:fromLayer:` (page 79)
Converts the point from the specified layer's coordinate system to the receiver's coordinate system.

– `convertPoint:toLayer:` (page 80)
Converts the point from the receiver's coordinate system to the specified layer's coordinate system.

– `convertRect:fromLayer:` (page 80)
Converts the rectangle from the specified layer's coordinate system to the receiver's coordinate system.

– `convertRect:toLayer:` (page 81)
Converts the rectangle from the receiver's coordinate system to the specified layer's coordinate system.

– `convertTime:fromLayer:` (page 82)
Converts the time interval from the specified layer's time space to the receiver's time space.

– `convertTime:toLayer:` (page 82)
Converts the time interval from the receiver's time space to the specified layer's time space

Hit Testing

– `hitTest:` (page 84)
Returns the farthest descendant of the receiver in the layer hierarchy (including itself) that contains the specified point.

– `containsPoint:` (page 78)
Returns whether the receiver contains a specified point.

Scrolling

`visibleRect` (page 71) *property*
The visible region of the layer in its own coordinate space. (read-only)

– `scrollPoint:` (page 93)
Initiates a scroll in the layer's closest ancestor scroll layer so that the specified point lies at the origin of the scroll layer.

– `scrollRectToVisible:` (page 93)
Initiates a scroll in the layer's closest ancestor scroll layer so that the specified rectangle becomes visible.

Identifying the Layer

`name` (page 62) *property*
The name of the receiver.

Key-Value Coding Extensions

– `shouldArchiveValueForKey:` (page 96)
Returns a Boolean indicating whether the value of the specified key should be archived.

+ `defaultValueForKey:` (page 72)
Specifies the default value associated with the specified key.

Properties

actions

A dictionary containing layer actions.

```
@property(copy) NSDictionary *actions
```

Discussion
The default value of this property is `nil`. You can use this dictionary to store custom actions for your layer. The contents of this dictionary searched as part of the standard implementation of the `actionForKey:` (page 74) method.

Availability
Available in iOS 2.0 and later.

See Also
– `actionForKey:` (page 74)

Declared in
`CALayer.h`

anchorPoint

Defines the anchor point of the layer's bounds rectangle. Animatable.

```
@property CGPoint anchorPoint
```

Discussion
You specify the value for this property using the unit coordinate space. The default value of this property is (0.5, 0.5), which represents the center of the layer's bounds rectangle. All geometric manipulations to the view occur about the specified point. For example, applying a rotation transform to a layer with the default anchor point causes the layer to rotate around its center. Changing the anchor point to a different location would cause the layer to rotate around that new point.

For more information about the relationship between the `frame` (page 58), `bounds` (page 50), `anchorPoint` (page 47) and `position` (page 64) properties, see *Core Animation Programming Guide*.

Availability
Available in iOS 2.0 and later.

See Also
`@property position` (page 64)

Related Sample Code
Handling Touches Using Responder Methods and Gesture Recognizers
MTAudioProcessingTap Audio Processor
oalTouch
UIKit Dynamics Catalog

Declared in
`CALayer.h`

anchorPointZ

The anchor point for the layer's position along the z axis. Animatable.

```
@property CGFloat anchorPointZ
```

Discussion
This property specifies the anchor point on the z axis around which geometric manipulations occur. The point is expressed as a distance (measured in points) along the z axis. The default value of this property is `0`.

Availability
Available in iOS 3.0 and later.

See Also
`@property anchorPoint` (page 47)

Declared in
`CALayer.h`

backgroundColor

The background color of the receiver. Animatable.

```
@property CGColorRef backgroundColor
```

Discussion
The default value of this property is `nil`.

The value of this property is retained using the Core Foundation retain/release semantics. This behavior occurs despite the fact that the property declaration appears to use the default assign semantics for object retention.

Availability
Available in iOS 2.0 and later.

Related Sample Code
AVLoupe

StitchedStreamPlayer

Declared in
`CALayer.h`

backgroundFilters

An array of Core Image filters to apply to the content immediately behind the layer. Animatable.

```
@property(copy) NSArray *backgroundFilters
```

Discussion
Background filters affect the content behind the layer that shows through into the layer itself. Typically this content belongs to the superlayer that acts as the parent of the layer. These filters do not affect the content of the layer itself, including the layer's background color and border. They also do not affect content outside of the layer's bounds.

The default value of this property is `nil`.

Changing the inputs of the `CIFilter` object directly after it is attached to the layer causes undefined behavior. In OS X, it is possible to modify filter parameters after attaching them to the layer but you must use the layer's `setValue:forKeyPath:` method to do so. In addition, you must assign a name to the filter so that you can identify it in the array. For example, to change the `inputScale` parameter of the filter, you could use code similar to the following:

```
CIFilter *filter = ...;
CALayer *layer = ...;


filter.name = @"myFilter";
layer.backgroundFilters = [NSArray arrayWithObject:filter];
[layer setValue:[NSNumber numberWithInt:1]
forKeyPath:@"backgroundFilters.myFilter.inputScale"];
```

Special Considerations
This property is not supported on layers in iOS.

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

borderColor

The color of the layer's border. Animatable.

```
@property CGColorRef borderColor
```

Discussion
The default value of this property is an opaque black color.

The value of this property is retained using the Core Foundation retain/release semantics. This behavior occurs despite the fact that the property declaration appears to use the default assign semantics for object retention.

Availability
Available in iOS 3.0 and later.

Declared in
`CALayer.h`

borderWidth

The width of the layer's border. Animatable.

```
@property CGFloat borderWidth
```

Discussion
When this value is greater than 0.0, the layer draws a border using the current `borderColor` (page 49) value. The border is drawn inset from the receiver's bounds by the value specified in this property. It is composited above the receiver's contents and sublayers and includes the effects of the `cornerRadius` (page 55) property.

The default value of this property is `0.0`.

Availability
Available in iOS 3.0 and later.

Declared in
`CALayer.h`

bounds

The layer's bounds rectangle. Animatable.

```
@property CGRect bounds
```

Discussion
The bounds rectangle is the origin and size of the layer in its own coordinate space. When you create a new standalone layer, the default value for this property is an empty rectangle, which you must change before using the layer. The values of each coordinate in the rectangle are measured in points.

For more information about the relationship between the `frame` (page 58), `bounds` (page 50), `anchorPoint` (page 47) and `position` (page 64) properties, see *Core Animation Programming Guide*.

Availability

Available in iOS 2.0 and later.

Related Sample Code

AVLoupe

AVSimpleEditoriOS

MotionGraphs

MTAudioProcessingTap Audio Processor

UIKit Dynamics Catalog

Declared in

`CALayer.h`

compositingFilter

A CoreImage filter used to composite the layer and the content behind it. Animatable.

```
@property(retain) id compositingFilter
```

Discussion

The default value of this property is `nil`, which causes the layer to use source-over compositing.

In OS X, it is possible to modify the filter's parameters after attaching it to the layer but you must use the layer's `setValue:forKeyPath:` method to do so. For example, to change the `inputScale` parameter of the filter, you could use code similar to the following:

```
CIFilter *filter = ...;
CALayer *layer = ...;

layer.compositingFilter = filter;
[layer setValue:[NSNumber numberWithInt:1]
forKeyPath:@"compositingFilter.inputScale"];
```

Changing the inputs of the `CIFilter` object directly after it is attached to the layer causes undefined behavior.

Special Considerations

This property is not supported on layers in iOS.

Availability
Available in iOS 2.0 and later.

See Also
`@property backgroundFilters` (page 49)

Declared in
`CALayer.h`

contents

An object that provides the contents of the layer. Animatable.

```
@property(retain) id contents
```

Discussion
The default value of this property is `nil`.

If you are using the layer to display a static image, you can set this property to the `CGImageRef` containing the image you want to display. (In OS X 10.6 and later, you can also set the property to an `NSImage` object.) Assigning a value to this property causes the layer to use your image rather than create a separate backing store.

If the layer object is tied to a view object, you should avoid setting the contents of this property directly. The interplay between views and layers usually results in the view replacing the contents of this property during a subsequent update.

Availability
Available in iOS 2.0 and later.

See Also
`@property contentsRect` (page 54)

Related Sample Code
MTAudioProcessingTap Audio Processor
oalTouch
UIKit Dynamics Catalog

Declared in
`CALayer.h`

contentsCenter

The rectangle that defines how the layer contents are scaled during a resizing operation. Animatable.

```
@property CGRect contentsCenter
```

Discussion

You can use this property to subdivide the layer's content into a 3x3 grid. The value in this property specifies the location and size of the center rectangle in that grid. If the layer's `contentsGravity` (page 53) property is set to one of the resizing modes, resizing the layer causes scaling to occur differently in each rectangle of the grid. The center rectangle is stretched in both dimensions, the top-center and bottom-center rectangles are stretched only horizontally, the left-center and right-center rectangles are stretched only vertically, and the four corner rectangles are not stretched at all. Therefore, you can use this technique to implement stretchable backgrounds or images using a three-part or nine-part image.

The value in this property is set to the unit rectangle (0.0,0.0) (1.0,1.0) by default, which causes the entire image to scale in both dimensions. If you specify a rectangle that extends outside the unit rectangle, the result is undefined. The rectangle you specify is applied only after the `contentsRect` (page 54) property has been applied to the image.

Note: If the width or height of the rectangle in this property is very small or `0`, the value is implicitly changed to the width or height of a single source pixel centered at the specified location.

Availability

Available in iOS 3.0 and later.

See Also

`@property contentsRect` (page 54)
`@property contentsGravity` (page 53)
`@property contents` (page 52)

Declared in

`CALayer.h`

contentsGravity

A constant that specifies how the layer's contents are positioned or scaled within its bounds.

```
@property(copy) NSString *contentsGravity
```

Discussion
The possible values for this property are listed in `“Contents Gravity Values”` (page 98). The default value of this property is `kCAGravityResize` (page 99).

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

contentsRect

The rectangle, in the unit coordinate space, that defines the portion of the layer’s contents that should be used. Animatable.

```
@property CGRect contentsRect
```

Discussion
Defaults to the unit rectangle (0.0, 0.0, 1.0, 1.0).

If pixels outside the unit rectangle are requested, the edge pixels of the contents image will be extended outwards.

If an empty rectangle is provided, the results are undefined.

Availability
Available in iOS 2.0 and later.

See Also
`@property contents` (page 52)

Declared in
`CALayer.h`

contentsScale

The scale factor applied to the layer.

```
@property CGFloat contentsScale
```

Discussion

This value defines the mapping between the logical coordinate space of the layer (measured in points) and the physical coordinate space (measured in pixels). Higher scale factors indicate that each point in the layer is represented by more than one pixel at render time. For example, if the scale factor is `2.0` and the layer's bounds are 50 x 50 points, the size of the bitmap used to present the layer's content is 100 x 100 pixels.

The default value of this property is 1.0. For layers attached to a view, the view changes the scale factor automatically to a value that is appropriate for the current screen. For layers you create and manage yourself, you must set the value of this property yourself based on the resolution of the screen and the content you are providing. Core Animation uses the value you specify as a cue to determine how to render your content.

Availability

Available in iOS 4.0 and later.

Declared in

`CALayer.h`

cornerRadius

The radius to use when drawing rounded corners for the layer's background. Animatable.

```
@property CGFloat cornerRadius
```

Discussion

Setting the radius to a value greater than `0.0` causes the layer to begin drawing rounded corners on its background. By default, the corner radius does not apply to the image in the layer's `contents` (page 52) property; it applies only to the background color and border of the layer. However, setting the `masksToBounds` (page 61) property to `YES` causes the content to be clipped to the rounded corners.

The default value of this property is `0.0`.

Availability

Available in iOS 3.0 and later.

Related Sample Code
GeocoderDemo

Declared in

`CALayer.h`

delegate

The layer's delegate object.

```
@property(assign) id delegate
```

Discussion
You can use a delegate object to provide the layer's contents, handle the layout of any sublayers, and provide custom actions in response to layer-related changes. The object you assign to this property should implement one or more of the methods of the `CALayerDelegate` informal protocol. For more information about that protocol, see *CALayerDelegate Informal Protocol Reference*

In iOS, if the layer is associated with a `UIView` object, this property *must* be set to the view that owns the layer.

Availability
Available in iOS 2.0 and later.

Related Sample Code
MotionGraphs

Declared in
`CALayer.h`

doubleSided

A Boolean indicating whether the layer displays its content when facing away from the viewer. Animatable.

```
@property(getter=isDoubleSided) BOOL doubleSided
```

Discussion
When the value in this property is `NO`, the layer hides its content when it faces away from the viewer. The default value of this property is `YES`.

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

drawsAsynchronously

A Boolean indicating whether drawing commands are deferred and processed asynchronously in a background thread.

```
@property BOOL drawsAsynchronously
```

Discussion
When this property is set to `YES`, the graphics context used to draw the layer's contents queues drawing commands and executes them on a background thread rather than executing them synchronously. Performing these commands asynchronously can improve performance in some apps. However, you should always measure the actual performance benefits before enabling this capability.

The default value for this property is `NO`.

Availability
Available in iOS 6.0 and later.

Declared in
`CALayer.h`

edgeAntialiasingMask

A bitmask defining how the edges of the receiver are rasterized.

```
@property unsigned int edgeAntialiasingMask
```

Discussion
This property specifies which edges of the layer are antialiased and is a combination of the constants defined in "Edge Antialiasing Mask" (page 97). You can enable or disable antialiasing for each edge (top, left, bottom, right) separately. By default antialiasing is enabled for all edges.

Typically, you would use this property to disable antialiasing for edges that abut edges of other layers, to eliminate the seams that would otherwise occur.

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

filters

An array of Core Image filters to apply to the contents of the layer and its sublayers. Animatable.

```
@property(copy) NSArray *filters
```

Discussion
The filters you add to this property affect the content of the layer, including its border, filled background and sublayers. The default value of this property is `nil`.

Changing the inputs of the `CIFilter` object directly after it is attached to the layer causes undefined behavior. It is possible to modify filter parameters after attaching them to the layer but you must use the layer's `setValue:forKeyPath:` method to do so. In addition, you must assign a name to the filter so that you can identify it in the array. For example, to change the `inputScale` parameter of the filter, you could use code similar to the following:

```
CIFilter *filter = ...;
CALayer *layer = ...;

filter.name = @"myFilter";
layer.filters = [NSArray arrayWithObject:filter];
[layer setValue:[NSNumber numberWithInt:1]
forKeyPath:@"filters.myFilter.inputScale"];
```

Special Considerations

This property is not supported on layers in iOS.

Availability

Available in iOS 2.0 and later.

Declared in

`CALayer.h`

frame

The layer's frame rectangle.

```
@property CGRect frame
```

Discussion

The frame rectangle is position and size of the layer specified in the superlayer's coordinate space. For layers, the frame rectangle is a computed property that is derived from the values in the`bounds` (page 50), `anchorPoint` (page 47) and `position` (page 64) properties. When you assign a new value to this property, the layer changes its `position` (page 64) and `bounds` (page 50) properties to match the rectangle you specified. The values of each coordinate in the rectangle are measured in points.

For more information about the relationship between the `frame` (page 58), `bounds` (page 50), `anchorPoint` (page 47) and `position` (page 64) properties, see *Core Animation Programming Guide*.

Note: The `frame` property is not directly animatable. Instead you should animate the appropriate combination of the `bounds` (page 50), `anchorPoint` (page 47) and `position` (page 64) properties to achieve the desired result.

Availability
Available in iOS 2.0 and later.

Related Sample Code
AccelerometerGraph

AVLoupe

AVSimpleEditoriOS

MotionGraphs

oalTouch

Declared in
`CALayer.h`

geometryFlipped

A Boolean that indicates whether the geometry of the layer and its sublayers is flipped vertically.

```
@property(getter=isGeometryFlipped) BOOL geometryFlipped
```

Discussion
If the layer is providing the backing for a layer-backed view, the view is responsible for managing the value in this property. For standalone layers, this property controls whether geometry values for the layer are interpreted using the standard or flipped coordinate system. THe value of this property does not affect the rendering of the layer's content.

The default value of this property is `NO`.

Availability
Available in iOS 3.0 and later.

Declared in
`CALayer.h`

hidden

A Boolean indicating whether the layer is displayed. Animatable.

```
@property(getter=isHidden) BOOL hidden
```

Discussion

The default value of this property is `NO`.

Availability

Available in iOS 2.0 and later.

Related Sample Code

oalTouch

StitchedStreamPlayer

UIKit Dynamics Catalog

Declared in

`CALayer.h`

magnificationFilter

The filter used when increasing the size of the content.

```
@property(copy) NSString *magnificationFilter
```

Discussion

The possible values for this property are listed in "`Scaling Filters`" (page 100).

The default value of this property is `kCAFilterLinear` (page 101).

Availability

Available in iOS 2.0 and later.

Declared in

`CALayer.h`

mask

An optional layer whose alpha channel is used to mask the layer's content.

```
@property(retain) CALayer *mask
```

Discussion

The layer's alpha channel determines how much of the layer's content and background shows through. Fully or partially opaque pixels allow the underlying content to show through but fully transparent pixels block that content.

The default value of this property is nil `nil`. When configuring a mask, remember to set the size and position of the mask layer to ensure it is aligned properly with the layer it masks.

Special Considerations

The layer you assign to this property must not have a superlayer. If it does, the behavior is undefined.

Availability

Available in iOS 3.0 and later.

See Also

`@property masksToBounds` (page 61)

Related Sample Code
AVLoupe

Declared in

`CALayer.h`

masksToBounds

A Boolean indicating whether sublayers are clipped to the layer's bounds. Animatable.

```
@property BOOL masksToBounds
```

Discussion

When the value of this property is `YES`, Core Animation creates an implicit clipping mask that matches the bounds of the layer and includes any corner radius effects. If a value for the `mask` (page 60) property is also specified, the two masks are multiplied to get the final mask value.

The default value of this property is `NO`.

Availability

Available in iOS 2.0 and later.

Related Sample Code
GeocoderDemo

Declared in

`CALayer.h`

minificationFilter

The filter used when reducing the size of the content.

```
@property(copy) NSString *minificationFilter
```

Discussion

The possible values for this property are listed in "Scaling Filters" (page 100).

The default value of this property is `kCAFilterLinear` (page 101).

Availability

Available in iOS 2.0 and later.

See Also

`@property minificationFilterBias` (page 62)

Declared in

`CALayer.h`

minificationFilterBias

The bias factor used by the minification filter to determine the levels of detail.

```
@property float minificationFilterBias
```

Discussion

This value is used by the `minificationFilter` (page 61) when it is set to `kCAFilterTrilinear` (page 101).

The default value of this property is `0.0`.

Availability

Available in iOS 3.0 and later.

Declared in

`CALayer.h`

name

The name of the receiver.

```
@property(copy) NSString *name
```

Discussion

The layer name is used by some layout managers to identify a layer. The default value of this property is `nil`.

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

needsDisplayOnBoundsChange

A Boolean indicating whether the layer contents must be updated when its bounds rectangle changes.

```
@property BOOL needsDisplayOnBoundsChange
```

Discussion
When this property is set to `YES`, the layer automatically calls its `setNeedsDisplay` (page 94) method whenever its `bounds` (page 50) property changes. The default value of this property is `NO`.

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

opacity

The opacity of the receiver. Animatable.

```
@property float opacity
```

Discussion
The value of this property must be in the range `0.0` (transparent) to `1.0` (opaque). Values outside that range are clamped to the minimum or maximum. The default value of this property is `1.0`.

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

opaque

A Boolean value indicating whether the layer contains completely opaque content.

```
@property(getter=isOpaque) BOOL opaque
```

Discussion

The default value of this property is `NO`. If your app draws completely opaque content that fills the layer's bounds, setting this property to `YES` lets the system optimize the rendering behavior for the layer. Specifically, when the layer creates the backing store for your drawing commands, Core Animation omits the alpha channel of that backing store. Doing so can improve the performance of compositing operations. If you set the value of this property to `YES`, you must fill the layer's bounds with opaque content.

Setting this property affects only the backing store managed by Core Animation. If you assign an image with an alpha channel to the layer's `contents` (page 52) property, that image retains its alpha channel regardless of the value of this property.

Availability

Available in iOS 2.0 and later.

Related Sample Code

GLAirplay

GLGravity

Real-time Video Processing Using AVPlayerItemVideoOutput

RosyWriter

SpeakHere

Declared in

`CALayer.h`

position

The layer's position in its superlayer's coordinate space. Animatable.

```
@property CGPoint position
```

Discussion

The value of this property is specified in points and is always specified relative to the value in the `anchorPoint` (page 47) property. For new standalone layers, the default position is set to (0.0, 0.0). Changing the `frame` (page 58) property also updates the value in this property.

For more information about the relationship between the `frame` (page 58), `bounds` (page 50), `anchorPoint` (page 47) and `position` (page 64) properties, see *Core Animation Programming Guide*.

Availability

Available in iOS 2.0 and later.

See Also
`@property anchorPoint` (page 47)

Related Sample Code
AccelerometerGraph

AVSimpleEditoriOS

MotionGraphs

MTAudioProcessingTap Audio Processor

oalTouch

Declared in
`CALayer.h`

rasterizationScale

The scale at which to rasterize content, relative to the coordinate space of the layer. Animatable

```
@property CGFloat rasterizationScale
```

Discussion
When the value in the `shouldRasterize` (page 68) property is `YES`, the layer uses this property to determine whether to scale the rasterized content (and by how much). The default value of this property is `1.0`, which indicates that the layer should be rasterized at its current size. Larger values magnify the content and smaller values shrink it.

Availability
Available in iOS 3.2 and later.

Declared in
`CALayer.h`

shadowColor

The color of the layer's shadow. Animatable.

```
@property CGColorRef shadowColor
```

Discussion
The default value of this property is an opaque black color.

The value of this property is retained using the Core Foundation retain/release semantics. This behavior occurs despite the fact that the property declaration appears to use the default assign semantics for object retention.

Availability

Available in iOS 3.2 and later.

Related Sample Code
MTAudioProcessingTap Audio Processor

Declared in

`CALayer.h`

shadowOffset

The offset (in points) of the layer's shadow. Animatable.

```
@property CGSize shadowOffset
```

Discussion

The default value of this property is (`0.0`, `–3.0`).

Availability

Available in iOS 3.2 and later.

Related Sample Code
MTAudioProcessingTap Audio Processor

Declared in

`CALayer.h`

shadowOpacity

The opacity of the layer's shadow. Animatable.

```
@property float shadowOpacity
```

Discussion

The value in this property must be in the range `0.0` (transparent) to `1.0` (opaque). The default value of this property is `0.0`.

Availability

Available in iOS 3.2 and later.

Related Sample Code
AVSimpleEditoriOS

MTAudioProcessingTap Audio Processor

Declared in
`CALayer.h`

shadowPath

The shape of the layer’s shadow. Animatable.

```
@property CGPathRef shadowPath
```

Discussion
The default value of this property is `nil`, which causes the layer to use a standard shadow shape. If you specify a value for this property, the layer creates its shadow using the specified path instead of the layer’s composited alpha channel. The path you provide defines the outline of the shadow. It is filled using the non-zero winding rule and the current shadow color, opacity, and blur radius.

Unlike most animatable properties, this property (as with all `CGPathRef` animatable properties) does not support implicit animation. However, the path object may be animated using any of the concrete subclasses of `CAPropertyAnimation`. Paths will interpolate as a linear blend of the "on-line" points; "off-line" points may be interpolated non-linearly (to preserve continuity of the curve's derivative). If the two paths have a different number of control points or segments, the results are undefined. If the path extends outside the layer bounds it will not automatically be clipped to the layer, only if the normal layer masking rules cause that.

Specifying an explicit path usually improves rendering performance.

The value of this property is retained using the Core Foundation retain/release semantics. This behavior occurs despite the fact that the property declaration appears to use the default assign semantics for object retention.

Availability
Available in iOS 3.2 and later.

Declared in
`CALayer.h`

shadowRadius

The blur radius (in points) used to render the layer’s shadow. Animatable.

```
@property CGFloat shadowRadius
```

Discussion
You specify the radius The default value of this property is 3.0.

Availability
Available in iOS 3.2 and later.

Related Sample Code
MTAudioProcessingTap Audio Processor

Declared in
`CALayer.h`

shouldRasterize

A Boolean that indicates whether the layer is rendered as a bitmap before compositing. Animatable

```
@property BOOL shouldRasterize
```

Discussion
When the value of this property is `YES`, the layer is rendered as a bitmap in its local coordinate space and then composited to the destination with any other content. Shadow effects and any filters in the `filters` (page 57) property are rasterized and included in the bitmap. However, the current opacity of the layer is not rasterized. If the rasterized bitmap requires scaling during compositing, the filters in the `minificationFilter` (page 61) and `magnificationFilter` (page 60) properties are applied as needed.

When the value of this property is `NO`, the layer is composited directly into the destination whenever possible. The layer may still be rasterized prior to compositing if certain features of the compositing model (such as the inclusion of filters) require it.

The default value of this property is `NO`.

Availability
Available in iOS 3.2 and later.

Related Sample Code
UnwindSegue

Declared in
`CALayer.h`

style

An optional dictionary used to store property values that aren't explicitly defined by the layer.

```
@property(copy) NSDictionary *style
```

Discussion

This dictionary may in turn have a `style` key, forming a hierarchy of default values. In the case of hierarchical style dictionaries the shallowest value for a property is used. For example, the value for "style.someValue" takes precedence over "style.style.someValue".

If the style dictionary does not define a value for an attribute, the receiver's `defaultValueForKey:` (page 72) method is called. The default value of this property is `nil`.

The style dictionary is not consulted for the following keys: `bounds`, `frame`.

Warning: If the style dictionary or any of its ancestors are modified, the values of the layer's properties are undefined until the `style` property is reset.

Availability

Available in iOS 2.0 and later.

Declared in

`CALayer.h`

sublayers

An array containing the layer's sublayers.

```
@property(copy) NSArray *sublayers
```

Discussion

The sublayers are listed in back to front order. The default value of this property is `nil`.

Special Considerations

When setting the `sublayers` property to an array populated with layer objects, each layer in the array must not already have a superlayer—that is, its `superlayer` (page 70) property must currently be `nil`.

Availability

Available in iOS 2.0 and later.

Related Sample Code
AVSimpleEditoriOS

Declared in

`CALayer.h`

sublayerTransform

Specifies the transform to apply to sublayers when rendering. Animatable.

```
@property CATransform3D sublayerTransform
```

Discussion

You typically use this property to add perspective and other viewing effects to embedded layers. You add perspective by setting the sublayer transform to the desired projection matrix. The default value of this property is the identity transform.

Availability

Available in iOS 2.0 and later.

Declared in

`CALayer.h`

superlayer

The superlayer of the layer. (read-only)

```
@property(readonly) CALayer *superlayer
```

Discussion

The superlayer manages the layout of its sublayers.

Availability

Available in iOS 2.0 and later.

Declared in

`CALayer.h`

transform

The transform applied to the layer's contents. Animatable.

```
@property CATransform3D transform
```

Discussion

This property is set to the identity transform by default. Any transformations you apply to the layer occur relative to the layer's anchor point.

Availability
Available in iOS 2.0 and later.

Related Sample Code
oalTouch

Declared in
`CALayer.h`

visibleRect

The visible region of the layer in its own coordinate space. (read-only)

```
@property(readonly) CGRect visibleRect
```

Discussion
The visible region is the area not clipped by the containing scroll layer.

Availability
Available in iOS 2.0 and later.

Declared in
`CAScrollLayer.h`

zPosition

The layer's position on the z axis. Animatable.

```
@property CGFloat zPosition
```

Discussion
The default value of this property is `0`. Changing the value of this property changes the the front-to-back ordering of layers onscreen. This can affect the visibility of layers whose frame rectangles overlap.

The value of this property is measured in points.

Availability
Available in iOS 2.0 and later.

Related Sample Code
PhotosByLocation

Declared in
`CALayer.h`

Class Methods

defaultActionForKey:

Returns the default action for the current class.

```
+ (id < CAAction >)defaultActionForKey:(NSString *)key
```

Parameters

`key`

The identifier of the action.

Return Value

Returns a suitable action object for the given key or `nil` of no action object was associated with that key.

Discussion

Classes that want to provide default actions can override this method and use it to return those actions.

Availability

Available in iOS 2.0 and later.

See Also

– `actionForKey:` (page 74)

Declared in

`CALayer.h`

defaultValueForKey:

Specifies the default value associated with the specified key.

```
+ (id)defaultValueForKey:(NSString *)key
```

Parameters

`key`

The name of one of the receiver's properties.

Return Value

The default value for the named property. Returns `nil` if no default value has been set.

Discussion

If you define custom properties for a layer but do not set a value, this method returns a suitable "zero" default value based on the expected value of the `key`. For example, if the value for `key` is a `CGSize` struct, the method returns a size struct containing (0.0,0.0) wrapped in an `NSValue` object. For a `CGRect` an empty rectangle is returned. For `CGAffineTransform` and `CATransform3D` (page 101), the appropriate identity matrix is returned.

Special Considerations

If `key` is not a known for property of the class, the result of the method is undefined.

Availability

Available in iOS 2.0 and later.

Declared in

`CALayer.h`

layer

Creates and returns an instance of the layer object.

```
+ (id)layer
```

Return Value

The initialized layer object or `nil` if initialization was not successful.

Discussion

If you subclass `CALayer`, you may override this method and use it to provide an instance of your specific subclass.

Availability

Available in iOS 2.0 and later.

Related Sample Code

AVLoupe

AVSimpleEditoriOS

iAdInterstitialSuite

MTAudioProcessingTap Audio Processor

oalTouch

Declared in

`CALayer.h`

needsDisplayForKey:

Returns a Boolean indicating whether changes to the specified key require the layer to be redisplayed.

```
+ (BOOL)needsDisplayForKey:(NSString *)key
```

Parameters

`key`

A string that specifies an attribute of the layer.

Return Value

`YES` if the layer requires a redisplay.

Discussion

Subclasses can override this method and return `YES` if the layer should be redisplayed when the value of the specified attribute changes. Animations changing the value of the attribute also trigger redisplay.

The default implementation of this method returns `NO`.

Availability

Available in iOS 3.0 and later.

See Also

`+ defaultActionForKey:` (page 72)
`+ defaultValueForKey:` (page 72)

Declared in

`CALayer.h`

Instance Methods

actionForKey:

Returns the action object assigned to the specified key.

```
- (id < CAAction >)actionForKey:(NSString *)key
```

Parameters

`key`

The identifier of the action.

Return Value

Returns the object that provides the action for `key`. The object must implement the `CAAction` protocol.

Discussion

This method searches for the given action object of the layer. Actions define dynamic behaviors for a layer. For example, the animatable properties of a layer typically have corresponding action objects to initiate the actual animations. When that property changes, the layer looks for the action object associated with the property name and executes it. You can also associate custom action objects with your layer to implement app-specific actions.

This method searches for the layer's associated actions in the following order:

1. If the layer has a delegate and that delegate implements the `actionForLayer:forKey:` method, the layer calls that method. The delegate must do one of the following:
   - Return the action object for the given key.
   - Return `nil` if it does not handle the action.
   - Return the `NSNull` object if it does not handle the action and the search should be terminated.
2. The layer looks in the layer's `actions` (page 46) dictionary.
3. The layer looks in the `style` (page 68) dictionary for an actions dictionary that contains the key.
4. The layer calls its `defaultActionForKey:` (page 72) method to look for any class-defined actions.
5. The layer looks for any implicit actions defined by Core Animation.

When an action object is invoked it receives three parameters: the name of the event, the object on which the event happened (the layer), and a dictionary of named arguments specific to each event kind.

Availability

Available in iOS 2.0 and later.

See Also

`– Accessing the Layer's Filters` (page 41)
`@property actions` (page 46)
`@property style` (page 68)
`+ defaultActionForKey:` (page 72)

Related Sample Code
iAdInterstitialSuite

Declared in

`CALayer.h`

addAnimation:forKey:

Add the specified animation object to the layer's render tree.

```
- (void)addAnimation:(CAAnimation *)anim forKey:(NSString *)key
```

Parameters

`anim`

The animation to be added to the render tree. This object is copied by the render tree, not referenced. Therefore, subsequent modifications to the object are not propagated into the render tree.

`key`

A string that identifies the animation. Only one animation per unique key is added to the layer. The special key `kCATransition` (page 97) is automatically used for transition animations. You may specify `nil` for this parameter.

Discussion

If the `duration` property of the animation is zero or negative, the duration is changed to the current value of the `kCATransactionAnimationDuration` (page 139) transaction property (if set) or to the default value of `0.25` seconds.

Availability

Available in iOS 2.0 and later.

Related Sample Code
MoveMe

Declared in

`CALayer.h`

addSublayer:

Appends the layer to the layer's list of sublayers.

```
- (void)addSublayer:(CALayer *)aLayer
```

Parameters

`aLayer`

The layer to be added.

Discussion

If the array in the sublayers property is `nil`, calling this method creates an array for that property and adds the specified layer to it.

Availability
Available in iOS 2.0 and later.

Related Sample Code
AVLoupe

AVSimpleEditoriOS

MTAudioProcessingTap Audio Processor

SquareCam

StopNGo for iOS

Declared in
`CALayer.h`

affineTransform

Returns an affine version of the layer's transform.

```
- (CGAffineTransform)affineTransform
```

Return Value
The affine transform structure that corresponds to the value in the layer's `transform` (page 70) property.

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

animationForKey:

Returns the animation object with the specified identifier.

```
- (CAAnimation *)animationForKey:(NSString *)key
```

Parameters
`key`
A string that specifies the identifier of the animation. This string corresponds to the identifier string you passed to the `addAnimation:forKey:` (page 76) method.

Return Value
The animation object matching the identifier, or `nil` if no such animation exists.

Discussion
You can use this string to retrieve animation objects already associated with a layer. However, you must not modify any properties of the returned object. Doing so will result in undefined behavior.

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

animationKeys

Returns an array of strings that identify the animations currently attached to the layer.

```
- (NSArray *)animationKeys
```

Return Value
An array of `NSString` objects identifying the current animations.

Discussion
The order of the array matches the order in which animations will be applied to the layer.

Availability
Available in iOS 3.0 and later.

Declared in
`CALayer.h`

containsPoint:

Returns whether the receiver contains a specified point.

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

Parameters
`thePoint`
    A point in the receiver's coordinate system.

Return Value
`YES` if the bounds of the layer contains the point.

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

contentsAreFlipped

Returns a Boolean indicating whether the layer content is implicitly flipped when rendered.

```
- (BOOL)contentsAreFlipped
```

Return Value
`YES` if the layer contents are implicitly flipped when rendered or `NO` if they are not. This method returns `NO` by default.

Discussion
This method provides information about whether the layer's contents are being flipped during drawing. You should not attempt to override this method and return a different value.

If the layer needs to flip its content, it returns `YES` from this method and applies a y-flip transform to the graphics context before passing it to the layer's `drawInContext:` (page 84) method. Similarly, the layer converts any rectangles passed to its `setNeedsDisplayInRect:` (page 95) into the flipped coordinate space.

Availability
Available in iOS 3.0 and later.

Declared in
`CALayer.h`

convertPoint:fromLayer:

Converts the point from the specified layer's coordinate system to the receiver's coordinate system.

```
- (CGPoint)convertPoint:(CGPoint)aPoint fromLayer:(CALayer *)layer
```

Parameters
`aPoint`
: A point specifying a location in the coordinate system of `layer`.

`layer`
: The layer with `aPoint` in its coordinate system. The receiver and `layer` and must share a common parent layer. This parameter may be `nil`.

Return Value

The point converted to the receiver's coordinate system.

Discussion

If you specify `nil` for the `layer` parameter, this method returns the original point.

Availability

Available in iOS 2.0 and later.

Declared in

`CALayer.h`

convertPoint:toLayer:

Converts the point from the receiver's coordinate system to the specified layer's coordinate system.

```
- (CGPoint)convertPoint:(CGPoint)aPoint toLayer:(CALayer *)layer
```

Parameters

`aPoint`

A point specifying a location in the coordinate system of `layer`.

`layer`

The layer into whose coordinate system `aPoint` is to be converted. The receiver and `layer` must share a common parent layer. This parameter may be `nil`.

Return Value

The point converted to the coordinate system of `layer`.

Discussion

If you specify `nil` for the `layer` parameter, this method returns the original point.

Availability

Available in iOS 2.0 and later.

Declared in

`CALayer.h`

convertRect:fromLayer:

Converts the rectangle from the specified layer's coordinate system to the receiver's coordinate system.

```
- (CGRect)convertRect:(CGRect)aRect fromLayer:(CALayer *)layer
```

Parameters

`aRect`
: A point specifying a location in the coordinate system of `layer`.

`layer`
: The layer with `aRect` in its coordinate system. The receiver and `layer` and must share a common parent layer. This parameter may be `nil`.

Return Value

The rectangle converted to the receiver's coordinate system.

Discussion

If you specify `nil` for the `layer` parameter, this method returns the original rect.

Availability

Available in iOS 2.0 and later.

Declared in

`CALayer.h`

convertRect:toLayer:

Converts the rectangle from the receiver's coordinate system to the specified layer's coordinate system.

```
- (CGRect)convertRect:(CGRect)aRect toLayer:(CALayer *)layer
```

Parameters

`aRect`
: A point specifying a location in the coordinate system of `layer`.

`layer`
: The layer into whose coordinate system `aRect` is to be converted. The receiver and `layer` and must share a common parent layer. This parameter may be `nil`.

Return Value

The rectangle converted to the coordinate system of `layer`.

Discussion

If you specify `nil` for the `layer` parameter, this method returns the original rect.

Availability

Available in iOS 2.0 and later.

Declared in
`CALayer.h`

convertTime:fromLayer:

Converts the time interval from the specified layer's time space to the receiver's time space.

```
- (CFTimeInterval)convertTime:(CFTimeInterval)timeInterval fromLayer:(CALayer *)layer
```

Parameters

`timeInterval`
A point specifying a location in the coordinate system of `layer`.

`layer`
The layer with `timeInterval` in its time space. The receiver and `layer` and must share a common parent layer.

Return Value
The time interval converted to the receiver's time space.

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

convertTime:toLayer:

Converts the time interval from the receiver's time space to the specified layer's time space

```
- (CFTimeInterval)convertTime:(CFTimeInterval)timeInterval toLayer:(CALayer *)layer
```

Parameters

`timeInterval`
A point specifying a location in the coordinate system of `layer`.

`layer`
The layer into whose time space `timeInterval` is to be converted. The receiver and `layer` and must share a common parent layer.

Return Value
The time interval converted to the time space of `layer`.

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

display

Reloads the content of this layer.

```
- (void)display
```

Discussion
Do not call this method directly. The layer calls this method at appropriate times to update the layer's content. If the layer has a delegate object, this method attempts to call the delegate's `displayLayer:` method, which the delegate can use to update the layer's contents. If the delegate does not implement the `displayLayer:` method, this method creates a backing store and calls the layer's `drawInContext:` (page 84) method to fill that backing store with content. The new backing store replaces the previous contents of the layer.

Subclasses can override this method and use it to set the layer's `contents` (page 52) property directly. You might do this if your custom layer subclass handles layer updates differently.

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

displayIfNeeded

Initiates the update process for a layer if it is currently marked as needing an update.

```
- (void)displayIfNeeded
```

Discussion
You can call this method as needed to force an update to your layer's contents outside of the normal update cycle. Doing so is generally not needed, though. The preferred way to update a layer is to call `setNeedsDisplay` (page 94) and let the system update the layer during the next cycle.

Availability
Available in iOS 3.0 and later.

See Also

– `needsDisplay` (page 89)

Declared in

`CALayer.h`

drawInContext:

Draws the layer's content using the specified graphics context.

```
- (void)drawInContext:(CGContextRef)ctx
```

Parameters

`ctx`

The graphics context in which to draw the content. The context may be clipped to protect valid layer content. Subclasses that wish to find the actual region to draw can call `CGContextGetClipBoundingBox`.

Discussion

The default implementation of this method does not doing any drawing itself. If the layer's delegate implements the `drawLayer:inContext:` method, that method is called to do the actual drawing.

Subclasses can override this method and use it to draw the layer's content. When drawing, all coordinates should be specified in points in the logical coordinate space.

Availability

Available in iOS 2.0 and later.

Declared in

`CALayer.h`

hitTest:

Returns the farthest descendant of the receiver in the layer hierarchy (including itself) that contains the specified point.

```
- (CALayer *)hitTest:(CGPoint)thePoint
```

Parameters

`thePoint`

A point in the coordinate system of the receiver's superlayer.

Return Value

The layer that contains `thePoint` or `nil` if the point lies outside the receiver's bounds rectangle.

Availability

Available in iOS 2.0 and later.

Declared in

`CALayer.h`

init

Returns an initialized `CALayer` object.

```
- (id)init
```

Return Value

An initialized `CALayer` object.

Discussion

This is the designated initializer for layer objects that are not in the presentation layer.

Availability

Available in iOS 2.0 and later.

See Also

`+ layer` (page 73)

Declared in

`CALayer.h`

initWithLayer:

Override to copy or initialize custom fields of the specified layer.

```
- (id)initWithLayer:(id)layer
```

Parameters

`layer`

The layer from which custom fields should be copied.

Return Value

A layer instance with any custom instance variables copied from `layer`.

Discussion

This initializer is used to create shadow copies of layers, for example, for the `presentationLayer` (page 90) method. Using this method in any other situation will produce undefined behavior. For example, do not use this method to initialize a new layer with an existing layer's content.

If you are implementing a custom layer subclass, you can override this method and use it to copy the values of instance variables into the new object. Subclasses should always invoke the superclass implementation.

This method is the designated initializer for layer objects in the presentation layer.

Availability

Available in iOS 2.0 and later.

Declared in

`CALayer.h`

insertSublayer:above:

Inserts the specified sublayer above a different sublayer that already belongs to the receiver.

```
- (void)insertSublayer:(CALayer *)aLayer above:(CALayer *)sublayer
```

Parameters

`aLayer`

The sublayer to be inserted into the current layer.

`sublayer`

An existing sublayer in the current layer. The layer in `aLayer` is inserted after this layer in the `sublayers` (page 69) array, and thus appears in front of it visually.

Special Considerations

If `sublayer` is not in the receiver's `sublayers` array, this method raises an exception.

Availability

Available in iOS 2.0 and later.

Declared in

`CALayer.h`

insertSublayer:atIndex:

Inserts the specified layer into the receiver's list of sublayers at the specified index.

```
- (void)insertSublayer:(CALayer *)aLayer atIndex:(unsigned)index
```

Parameters

`aLayer`
: The sublayer to be inserted into the current layer.

`index`
: The index at which to insert `aLayer`. This value must be a valid 0-based index into the `sublayers` (page 69) array.

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

insertSublayer:below:

Inserts the specified sublayer below a different sublayer that already belongs to the receiver.

```
- (void)insertSublayer:(CALayer *)aLayer below:(CALayer *)sublayer
```

Parameters

`aLayer`
: The sublayer to be inserted into the current layer.

`sublayer`
: An existing sublayer in the current layer. The layer in `aLayer` is inserted before this layer in the `sublayers` (page 69) array, and thus appears behind it visually.

Discussion
If `sublayer` is not in the receiver's `sublayers` array, this method raises an exception.

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

layoutIfNeeded

Recalculate the receiver's layout, if required.

```
- (void)layoutIfNeeded
```

Discussion

When this message is received, the layer’s super layers are traversed until a ancestor layer is found that does not require layout. Then layout is performed on the entire layer-tree beneath that ancestor.

Availability

Available in iOS 2.0 and later.

Declared in

`CALayer.h`

layoutSublayers

Tells the layer to update its layout.

```
- (void)layoutSublayers
```

Discussion

Subclasses can override this method and use it to implement their own layout algorithm. Your implementation must set the frame of each sublayer managed by the receiver.

The default implementation of this method calls the `layoutSublayersOfLayer:` method of the layer’s delegate object. If there is no delegate object, or the delegate does not implement that method, this method calls the `layoutSublayersOfLayer:` method of the object in the `layoutManager` property.

Availability

Available in iOS 2.0 and later.

Declared in

`CALayer.h`

modelLayer

Returns the model layer object associated with the receiver, if any.

```
- (id)modelLayer
```

Return Value

A layer instance representing the underlying model layer.

Discussion

Calling this method on a layer in the presentation tree returns the corresponding layer object in the model tree. This method returns a value only when a transaction involving changes to the presentation layer is in progress. If no transaction is in progress, the results of calling this method are undefined.

Availability

Available in iOS 2.0 and later.

Declared in

`CALayer.h`

needsDisplay

Returns a Boolean indicating whether the layer has been marked as needing an update.

```
- (BOOL)needsDisplay
```

Return Value

`YES` if the layer needs to be updated.

Availability

Available in iOS 3.0 and later.

Declared in

`CALayer.h`

needsLayout

Returns a Boolean indicating whether the layer has been marked as needing a layout update.

```
- (BOOL)needsLayout
```

Return Value

`YES` if the layer has been marked as requiring a layout update.

Availability

Available in iOS 3.0 and later.

See Also

– `setNeedsLayout` (page 95)

Declared in

`CALayer.h`

preferredFrameSize

Returns the preferred size of the layer in the coordinate space of its superlayer.

```
- (CGSize)preferredFrameSize
```

Return Value
The layer's preferred frame size.

Discussion
In OS X, the default implementation of this method calls the `preferredSizeOfLayer:` method of its layout manager—that is, the object in its `layoutManager` property. If that object does not exist or does not implement that method, this method returns the size of the layer's current `bounds` (page 50) rectangle mapped into the coordinate space of its `superlayer` (page 70).

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

presentationLayer

Returns a copy of the presentation layer object that represents the state of the layer as it currently appears onscreen.

```
- (id)presentationLayer
```

Return Value
A copy of the current presentation layer object.

Discussion
The layer object returned by this method provides a close approximation of the layer that is currently being displayed onscreen. While an animation is in progress, you can retrieve this object and use it to get the current values for those animations.

The `sublayers` (page 69), `mask` (page 60), and `superlayer` (page 70) properties of the returned layer return the corresponding objects from the presentation tree (not the model tree). This pattern also applies to any read-only layer methods. For example, the `hitTest:` (page 84) method of the returned object queries the layer objects in the presentation tree.

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

removeAllAnimations

Remove all animations attached to the layer.

```
- (void)removeAllAnimations
```

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

removeAnimationForKey:

Remove the animation object with the specified key.

```
- (void)removeAnimationForKey:(NSString *)key
```

Parameters
`key`
    The identifier of the animation to remove.

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

removeFromSuperlayer

Detaches the layer from its parent layer.

```
- (void)removeFromSuperlayer
```

Discussion
You can use this method to remove a layer (and all of its sublayers) from a layer hierarchy. This method updates both the superlayer's list of sublayers and sets this layer's `superlayer` (page 70) property to `nil`.

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

renderInContext:

Renders the receiver and its sublayers into the specified context.

```
- (void)renderInContext:(CGContextRef)ctx
```

Parameters

`ctx`
The graphics context to use to render the layer.

Discussion
This method renders directly from the layer tree, ignoring any animations added to the render tree. Renders in the coordinate space of the layer.

Important: The OS X v10.5 implementation of this method does not support the entire Core Animation composition model. `QCCompositionLayer`, `CAOpenGLLayer`, and `QTMovieLayer` layers are not rendered. Additionally, layers that use 3D transforms are not rendered, nor are layers that specify `backgroundFilters` (page 49), `filters` (page 57), `compositingFilter` (page 51), or a `mask` (page 60) values. Future versions of OS X may add support for rendering these layers and properties.

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

replaceSublayer:with:

Replaces the specified sublayer with a different layer object.

```
- (void)replaceSublayer:(CALayer *)oldLayer with:(CALayer *)newLayer
```

Parameters

`oldLayer`
The layer to be replaced.

`newLayer`
: The layer with which to replace `oldLayer`.

Discussion

If `oldLayer` is not in the receiver's `sublayers` (page 69) array, the behavior of this method is undefined.

Availability

Available in iOS 2.0 and later.

Declared in

`CALayer.h`

scrollPoint:

Initiates a scroll in the layer's closest ancestor scroll layer so that the specified point lies at the origin of the scroll layer.

```
- (void)scrollPoint:(CGPoint)thePoint
```

Parameters

`thePoint`
: The point in the current layer that should be scrolled into position.

Discussion

If the layer is not contained by a `CAScrollLayer` object, this method does nothing.

Availability

Available in iOS 2.0 and later.

Declared in

`CAScrollLayer.h`

scrollRectToVisible:

Initiates a scroll in the layer's closest ancestor scroll layer so that the specified rectangle becomes visible.

```
- (void)scrollRectToVisible:(CGRect)theRect
```

Parameters

`theRect`
: The rectangle to be made visible.

Discussion
If the layer is not contained by a `CAScrollLayer` object, this method does nothing.

Availability
Available in iOS 2.0 and later.

Declared in
`CAScrollLayer.h`

setAffineTransform:

Sets the layer's transform to the specified affine transform.

```
- (void)setAffineTransform:(CGAffineTransform)m
```

Parameters
`m`

The affine transform to use for the layer's transform.

Availability
Available in iOS 2.0 and later.

Related Sample Code
SquareCam

UIKit Dynamics Catalog

Declared in
`CALayer.h`

setNeedsDisplay

Marks the layer's contents as needing to be updated.

```
- (void)setNeedsDisplay
```

Discussion
Calling this method causes the layer to recache its content. This results in the layer potentially calling either the `displayLayer:` or `drawLayer:inContext:` method of its delegate. The existing content in the layer's `contents` (page 52) property is removed to make way for the new content.

Availability
Available in iOS 2.0 and later.

Related Sample Code
AccelerometerGraph

Declared in
`CALayer.h`

setNeedsDisplayInRect:

Marks the region within the specified rectangle as needing to be updated.

```
- (void)setNeedsDisplayInRect:(CGRect)theRect
```

Parameters

`theRect`
: The rectangular region of the layer to mark as invalid. You must specify this rectangle in the layer's own coordinate system.

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

setNeedsLayout

Invalidates the layer's layout and marks it as needing an update.

```
- (void)setNeedsLayout
```

Discussion
You can call this method to indicate that the layout of a layer's sublayers has changed and must be updated. The system typically calls this method automatically when the layer's bounds change or when sublayers are added or removed. In OS X, if your layer's `layoutManager` property contains an object that implements the `invalidateLayoutOfLayer:` method, that method is called too.

During the next update cycle, the system calls the `layoutSublayers` (page 88) method of any layers requiring layout updates.

Availability
Available in iOS 2.0 and later.

Declared in
`CALayer.h`

shouldArchiveValueForKey:

Returns a Boolean indicating whether the value of the specified key should be archived.

```
- (BOOL)shouldArchiveValueForKey:(NSString *)key
```

Parameters

`key`

The name of one of the receiver's properties.

Return Value

`YES` if the specified property should be archived or `NO` if it should not.

Discussion

The default implementation returns `YES`.

Availability

Available in iOS 4.0 and later.

Declared in

`CALayer.h`

Constants

Action Identifiers

These constants are the predefined action identifiers used by `actionForKey:` (page 74), `addAnimation:forKey:` (page 76), `defaultActionForKey:` (page 72), `removeAnimationForKey:` (page 91), `Accessing the Layer's Filters` (page 41), and the `CAAction` protocol method `runActionForKey:object:arguments:` (page 239).

```
NSString * const kCAOnOrderIn;
NSString * const kCAOnOrderOut;
NSString * const kCATransition;
```

Constants

`kCAOnOrderIn`

The identifier that represents the action taken when a layer becomes visible, either as a result being inserted into the visible layer hierarchy or the layer is no longer set as hidden.

Available in iOS 2.0 and later.

Declared in `CALayer.h`.

`kCAOnOrderOut`

The identifier that represents the action taken when the layer is removed from the layer hierarchy or is hidden.

Available in iOS 2.0 and later.

Declared in `CALayer.h`.

`kCATransition`

The identifier that represents a transition animation.

Available in iOS 2.0 and later.

Declared in `CALayer.h`.

Declared in

`CALayer.h`

Edge Antialiasing Mask

This mask is used by the `edgeAntialiasingMask` *(page 57) property.*

```
enum CAEdgeAntialiasingMask
{
   kCALayerLeftEdge    = 1U << 0,
   kCALayerRightEdge    = 1U << 1,
   kCALayerBottomEdge    = 1U << 2,
   kCALayerTopEdge    = 1U << 3,
};
```

Constants

`kCALayerLeftEdge`

Specifies that the left edge of the receiver's content should be antialiased.

Available in iOS 2.0 and later.

Declared in `CALayer.h`.

`kCALayerRightEdge`

Specifies that the right edge of the receiver's content should be antialiased.

Available in iOS 2.0 and later.

Declared in `CALayer.h`.

`kCALayerBottomEdge`

Specifies that the bottom edge of the receiver's content should be antialiased.

Available in iOS 2.0 and later.

Declared in `CALayer.h`.

`kCALayerTopEdge`

Specifies that the top edge of the receiver's content should be antialiased.

Available in iOS 2.0 and later.

Declared in `CALayer.h`.

Declared in
`CALayer.h`

Contents Gravity Values

The contents gravity constants specify the position of the content object when the layer bounds is larger than the bounds of the content object. The are used by the `contentsGravity` *(page 53) property.*

```
NSString * const kCAGravityCenter;
NSString * const kCAGravityTop;
NSString * const kCAGravityBottom;
NSString * const kCAGravityLeft;
NSString * const kCAGravityRight;
NSString * const kCAGravityTopLeft;
NSString * const kCAGravityTopRight;
NSString * const kCAGravityBottomLeft;
NSString * const kCAGravityBottomRight;
NSString * const kCAGravityResize;
NSString * const kCAGravityResizeAspect;
NSString * const kCAGravityResizeAspectFill;
```

Constants
`kCAGravityCenter`

The content is horizontally and vertically centered in the bounds rectangle.

Available in iOS 2.0 and later.

Declared in `CALayer.h`.

`kCAGravityTop`
: The content is horizontally centered at the top-edge of the bounds rectangle.

  Available in iOS 2.0 and later.

  Declared in `CALayer.h`.

`kCAGravityBottom`
: The content is horizontally centered at the bottom-edge of the bounds rectangle.

  Available in iOS 2.0 and later.

  Declared in `CALayer.h`.

`kCAGravityLeft`
: The content is vertically centered at the left-edge of the bounds rectangle.

  Available in iOS 2.0 and later.

  Declared in `CALayer.h`.

`kCAGravityRight`
: The content is vertically centered at the right-edge of the bounds rectangle.

  Available in iOS 2.0 and later.

  Declared in `CALayer.h`.

`kCAGravityTopLeft`
: The content is positioned in the top-left corner of the bounds rectangle.

  Available in iOS 2.0 and later.

  Declared in `CALayer.h`.

`kCAGravityTopRight`
: The content is positioned in the top-right corner of the bounds rectangle.

  Available in iOS 2.0 and later.

  Declared in `CALayer.h`.

`kCAGravityBottomLeft`
: The content is positioned in the bottom-left corner of the bounds rectangle.

  Available in iOS 2.0 and later.

  Declared in `CALayer.h`.

`kCAGravityBottomRight`
: The content is positioned in the bottom-right corner of the bounds rectangle.

  Available in iOS 2.0 and later.

  Declared in `CALayer.h`.

`kCAGravityResize`
: The content is resized to fit the entire bounds rectangle.

  Available in iOS 2.0 and later.

  Declared in `CALayer.h`.

`kCAGravityResizeAspect`
: The content is resized to fit the bounds rectangle, preserving the aspect of the content. If the content does not completely fill the bounds rectangle, the content is centered in the partial axis.

  Available in iOS 2.0 and later.

  Declared in `CALayer.h`.

`kCAGravityResizeAspectFill`
: The content is resized to completely fill the bounds rectangle, while still preserving the aspect of the content. The content is centered in the axis it exceeds.

  Available in iOS 2.0 and later.

  Declared in `CALayer.h`.

Declared in
`CALayer.h`

Identity Transform

Defines the identity transform matrix used by Core Animation.

```
const CATransform3D CATransform3DIdentity
```

Constants

`CATransform3DIdentity`
: The identity transform: [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1].

  Available in iOS 2.0 and later.

  Declared in `CATransform3D.h`.

Declared in
`CATransform3D.h`

Scaling Filters

These constants specify the scaling filters used by `magnificationFilter` (page 60) and `minificationFilter` (page 61).

```
NSString * const kCAFilterLinear;
NSString * const kCAFilterNearest;
NSString * const kCAFilterTrilinear;
```

Constants

`kCAFilterLinear`

Linear interpolation filter.

Available in iOS 2.0 and later.

Declared in `CALayer.h`.

`kCAFilterNearest`

Nearest neighbor interpolation filter.

Available in iOS 2.0 and later.

Declared in `CALayer.h`.

`kCAFilterTrilinear`

Trilinear minification filter. Enables mipmap generation. Some renderers may ignore this, or impose additional restrictions, such as source images requiring power-of-two dimensions..

Available in iOS 3.0 and later.

Declared in `CALayer.h`.

Declared in

`CALayer.h`

CATransform3D

Defines the standard transform matrix used throughout Core Animation.

```
struct CATransform3D
{
CGFloat m11, m12, m13, m14;
CGFloat m21, m22, m23, m24;
CGFloat m31, m32, m33, m34;
CGFloat m41, m42, m43, m44;
};
typedef struct CATransform3D CATransform3D;
```

Fields

`m11`

The entry at position 1,1 in the matrix.

`m12`

The entry at position 1,2 in the matrix.

`m13`

The entry at position 1,3 in the matrix.

`m14`
: The entry at position 1,4 in the matrix.

`m21`
: The entry at position 2,1 in the matrix.

`m22`
: The entry at position 2,2 in the matrix.

`m23`
: The entry at position 2,3 in the matrix.

`m24`
: The entry at position 2,4 in the matrix.

`m31`
: The entry at position 3,1 in the matrix.

`m32`
: The entry at position 3,2 in the matrix.

`m33`
: The entry at position 3,3 in the matrix.

`m34`
: The entry at position 3,4 in the matrix.

`m41`
: The entry at position 4,1 in the matrix.

`m42`
: The entry at position 4,2 in the matrix.

`m43`
: The entry at position 4,3 in the matrix.

`m44`
: The entry at position 4,4 in the matrix.

Discussion
The transform matrix is used to rotate, scale, translate, skew, and project the layer content. Functions are provided for creating, concatenating, and modifying CATransform3D data.

Availability
Available in iOS 2.0 and later.

Declared in
`CATransform3D.h`

CAMediaTimingFunction Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in iOS 2.0 and later.
Declared in	CAMediaTimingFunction.h
Companion guides	Core Animation Programming Guide Core Animation Cookbook
Related sample code	MoveMe

Overview

`CAMediaTimingFunction` represents one segment of a function that defines the pacing of an animation as a timing curve. The function maps an input time normalized to the range [0,1] to an output time also in the range [0,1].

Tasks

Creating Timing Functions

`+ functionWithName:` (page 105)
Creates and returns a new instance of `CAMediaTimingFunction` configured with the predefined timing function specified by `name`.

`+ functionWithControlPoints::::` (page 104)
Creates and returns a new instance of `CAMediaTimingFunction` timing function modeled as a cubic Bézier curve using the specified control points.

– `initWithControlPoints::::` (page 106)
  Returns an initialized timing function modeled as a cubic Bézier curve using the specified control points.

Accessing the Control Points

– `getControlPointAtIndex:values:` (page 105)
  Returns the control point for the specified index.

Class Methods

functionWithControlPoints::::

Creates and returns a new instance of `CAMediaTimingFunction` *timing function modeled as a cubic Bézier curve using the specified control points.*

```
+ (id)functionWithControlPoints:(float)c1x
:(float)c1y
:(float)c2x
:(float)c2y
```

Parameters

`c1x`
  A floating point number representing the x position of the c1 control point.

`c1y`
  A floating point number representing the y position of the c1 control point.

`c2x`
  A floating point number representing the x position of the c2 control point.

`c2y`
  A floating point number representing the y position of the c2 control point.

Return Value

A new instance of `CAMediaTimingFunction` with the timing function specified by the provided control points.

Discussion

The end points of the Bézier curve are automatically set to (0.0,0.0) and (1.0,1.0). The control points defining the Bézier curve are: [(0.0,0.0), (`c1x`,`c1y`), (`c2x`,`c2y`), (1.0,1.0)].

Availability
Available in iOS 2.0 and later.

Declared in
`CAMediaTimingFunction.h`

functionWithName:

Creates and returns a new instance of `CAMediaTimingFunction` configured with the predefined timing function specified by `name`.

```
+ (id)functionWithName:(NSString *)name
```

Parameters
`name`

The timing function to use as specified in “`Predefined Timing Functions`” (page 107).

Return Value
A new instance of `CAMediaTimingFunction` with the timing function specified by `name`.

Availability
Available in iOS 2.0 and later.

Related Sample Code
MoveMe

Declared in
`CAMediaTimingFunction.h`

Instance Methods

getControlPointAtIndex:values:

Returns the control point for the specified index.

```
- (void)getControlPointAtIndex:(size_t)index values:(float)ptr
```

Parameters
`index`

An integer specifying the index of the control point to return.

`ptr`

A pointer to an array that, upon return, will contain the x and y values of the specified point.

Discussion

The value of `index` must be between 0 and 3.

Availability

Available in iOS 2.0 and later.

Declared in

`CAMediaTimingFunction.h`

initWithControlPoints::::

Returns an initialized timing function modeled as a cubic Bézier curve using the specified control points.

```
- (id)initWithControlPoints:(float)c1x
:(float)c1y
:(float)c2x
:(float)c2y
```

Parameters

`c1x`

A floating point number representing the x position of the c1 control point.

`c1y`

A floating point number representing the y position of the c1 control point.

`c2x`

A floating point number representing the x position of the c2 control point.

`c2y`

A floating point number representing the y position of the c2 control point.

Return Value

An instance of `CAMediaTimingFunction` with the timing function specified by the provided control points.

Discussion

The end points of the Bézier curve are automatically set to (0.0,0.0) and (1.0,1.0). The control points defining the Bézier curve are: [(0.0,0.0), (`c1x`,`c1y`), (`c2x`,`c2y`), (1.0,1.0)].

Availability

Available in iOS 2.0 and later.

Declared in

`CAMediaTimingFunction.h`

Constants

Predefined Timing Functions

These constants are used to specify one of the predefined timing functions used by `functionWithName:` *(page 105).*

```
NSString * const kCAMediaTimingFunctionLinear;
NSString * const kCAMediaTimingFunctionEaseIn;
NSString * const kCAMediaTimingFunctionEaseOut;
NSString * const kCAMediaTimingFunctionEaseInEaseOut;
NSString * const kCAMediaTimingFunctionDefault;
```

Constants

`kCAMediaTimingFunctionLinear`

Specifies linear pacing. Linear pacing causes an animation to occur evenly over its duration.

Available in iOS 2.0 and later.

Declared in `CAMediaTimingFunction.h`.

`kCAMediaTimingFunctionEaseIn`

Specifies ease-in pacing. Ease-in pacing causes the animation to begin slowly, and then speed up as it progresses.

Available in iOS 2.0 and later.

Declared in `CAMediaTimingFunction.h`.

`kCAMediaTimingFunctionEaseOut`

Specifies ease-out pacing. An ease-out pacing causes the animation to begin quickly, and then slow as it completes.

Available in iOS 2.0 and later.

Declared in `CAMediaTimingFunction.h`.

`kCAMediaTimingFunctionEaseInEaseOut`

Specifies ease-in ease-out pacing. An ease-in ease-out animation begins slowly, accelerates through the middle of its duration, and then slows again before completing.

Available in iOS 2.0 and later.

Declared in `CAMediaTimingFunction.h`.

`kCAMediaTimingFunctionDefault`

Specifies the timing function used as the default by most animations. It approximates a Bézier timing function using the control points [(0.0,0.0), (0.25,0.1), (0.25,0.1), (1.0,1.0)]. By using this constant you ensure that your animations will use the current default timing.

Available in iOS 3.0 and later.

Declared in `CAMediaTimingFunction.h`.

CAPropertyAnimation Class Reference

Inherits from	CAAnimation : NSObject
Conforms to	NSCoding (CAAnimation) NSCopying (CAAnimation) CAAction (CAAnimation) CAMediaTiming (CAAnimation) NSObject (NSObject)
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in iOS 2.0 and later.
Declared in	CAAnimation.h
Companion guides	Core Animation Programming Guide Core Animation Cookbook

Overview

`CAPropertyAnimation` is an abstract subclass of `CAAnimation` for creating animations that manipulate the value of layer properties. The property is specified using a key path that is relative to the layer using the animation.

Tasks

Animated Key Path

`keyPath` (page 110) *property*
  Specifies the key path the receiver animates.

Property Value Calculation Behavior

`cumulative` (page 110) *property*
Determines if the value of the property is the value at the end of the previous repeat cycle, plus the value of the current repeat cycle.

`additive` (page 109) *property*
Determines if the value specified by the animation is added to the current render tree value to produce the new render tree value.

`valueFunction` (page 110) *property*
An optional value function that is applied to interpolated values.

Creating an Animation

`+ animationWithKeyPath:` (page 111)
Creates and returns an `CAPropertyAnimation` instance for the specified key path.

Properties

additive

Determines if the value specified by the animation is added to the current render tree value to produce the new render tree value.

```
@property(getter=isAdditive) BOOL additive
```

Discussion
If `YES`, the value specified by the animation will be added to the current render tree value of the property to produce the new render tree value. The addition function is type-dependent, e.g. for affine transforms the two matrices are concatenated. The default is `NO`.

Availability
Available in iOS 2.0 and later.

Declared in
`CAAnimation.h`

cumulative

Determines if the value of the property is the value at the end of the previous repeat cycle, plus the value of the current repeat cycle.

```
@property(getter=isCumulative) BOOL cumulative
```

Discussion

If `YES`, then the value of the property is the value at the end of the previous repeat cycle, plus the value of the current repeat cycle. If `NO`, the value of the property is simply the value calculated for the current repeat cycle. The default is `NO`.

Availability

Available in iOS 2.0 and later.

Declared in

`CAAnimation.h`

keyPath

Specifies the key path the receiver animates.

```
@property(copy) NSString *keyPath
```

Discussion

The key path is relative to the layer the receiver is attached to.

Availability

Available in iOS 2.0 and later.

Declared in

`CAAnimation.h`

valueFunction

An optional value function that is applied to interpolated values.

```
@property(retain) CAValueFunction *valueFunction
```

Discussion

If the `valueFunction` property is not `nil`, the function is applied to the values interpolated by the animation as they are applied to the presentation layer. Defaults to `nil`.

Availability

Available in iOS 3.0 and later.

Declared in

`CAAnimation.h`

Class Methods

animationWithKeyPath:

Creates and returns an `CAPropertyAnimation` instance for the specified key path.

```
+ (id)animationWithKeyPath:(NSString *)keyPath
```

Parameters

`keyPath`

The key path of the property to be animated.

Return Value

A new instance of `CAPropertyAnimation` with the key path set to `keyPath`.

Availability

Available in iOS 2.0 and later.

Related Sample Code
iAdInterstitialSuite

MoveMe

Declared in

`CAAnimation.h`

CAScrollLayer Class Reference

Inherits from	CALayer : NSObject
Conforms to	NSCoding (CALayer) CAMediaTiming (CALayer) NSObject (NSObject)
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in iOS 2.0 and later.
Declared in	CAScrollLayer.h
Companion guides	Core Animation Programming Guide Core Animation Cookbook

Overview

The `CAScrollLayer` class is a subclass of `CALayer` that simplifies displaying a portion of a layer. The extent of the scrollable area of the `CAScrollLayer` is defined by the layout of its sublayers. The visible portion of the layer content is set by specifying the origin as a point or a rectangular area of the contents to be displayed. `CAScrollLayer` does not provide keyboard or mouse event-handling, nor does it provide visible scrollers.

Tasks

Scrolling Constraints

`scrollMode` (page 113) *property*
Defines the axes in which the layer may be scrolled.

Scrolling the Layer

– `scrollToPoint:` (page 113)
  Changes the origin of the receiver to the specified point.

– `scrollToRect:` (page 114)
  Scroll the contents of the receiver to ensure that the rectangle is visible.

Properties

scrollMode

Defines the axes in which the layer may be scrolled.

```
@property(copy) NSString *scrollMode
```

Discussion
The possible values are described in "`Scroll Modes`" (page 114). The default is `kCAScrollBoth`.

Availability
Available in iOS 2.0 and later.

Declared in
`CAScrollLayer.h`

Instance Methods

scrollToPoint:

Changes the origin of the receiver to the specified point.

```
- (void)scrollToPoint:(CGPoint)thePoint
```

Parameters
`thePoint`
  The new origin.

Availability
Available in iOS 2.0 and later.

Declared in
`CAScrollLayer.h`

scrollToRect:

Scroll the contents of the receiver to ensure that the rectangle is visible.

```
- (void)scrollToRect:(CGRect)theRect
```

Parameters

`theRect`
The rectangle that should be visible.

Availability
Available in iOS 2.0 and later.

Declared in
`CAScrollLayer.h`

Constants

Scroll Modes

These constants describe the supported scroll modes used by the scrollMode (page 113) property.

```
NSString * const kCAScrollNone;
NSString * const kCAScrollVertically;
NSString * const kCAScrollHorizontally;
NSString * const kCAScrollBoth;
```

Constants

`kCAScrollNone`
The receiver is unable to scroll.
Available in iOS 2.0 and later.
Declared in `CAScrollLayer.h`.

`kCAScrollVertically`
The receiver is able to scroll vertically.
Available in iOS 2.0 and later.
Declared in `CAScrollLayer.h`.

`kCAScrollHorizontally`

The receiver is able to scroll horizontally.

Available in iOS 2.0 and later.

Declared in `CAScrollLayer.h`.

`kCAScrollBoth`

The receiver is able to scroll both horizontally and vertically.

Available in iOS 2.0 and later.

Declared in `CAScrollLayer.h`.

Declared in
`CAScrollLayer.h`

CATextLayer Class Reference

Inherits from	CALayer : NSObject
Conforms to	NSCoding (CALayer) CAMediaTiming (CALayer) NSObject (NSObject)
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in iOS 3.2 and later.
Declared in	CATextLayer.h
Companion guides	Core Animation Programming Guide Core Animation Cookbook
Related sample code	AVSimpleEditoriOS

Overview

The `CATextLayer` provides simple text layout and rendering of plain or attributed strings. The first line is aligned to the top of the layer.

Note: `CATextLayer` disables sub-pixel antialiasing when rendering text. Text can only be drawn using sub-pixel antialiasing when it is composited into an existing opaque background at the same time that it's rasterized. There is no way to draw text with sub-pixel antialiasing by itself, whether into an image or a layer, in advance of having the background pixels to weave the text pixels into. Setting the `opacity` property of the layer to YES does not change the rendering mode.

Note: In OS X, when a `CATextLayer` instance is positioned using the `CAConstraintLayoutManager` class the bounds of the layer is resized to fit the text content.

Tasks

Getting and Setting the Text

`string` (page 120) *property*
The text to be rendered by the receiver.

Text Visual Properties

`font` (page 118) *property*
The font used to render the receiver's text.

`fontSize` (page 119) *property*
The font size used to render the receiver's text. Animatable.

`foregroundColor` (page 119) *property*
The color used to render the receiver's text. Animatable.

Text Alignment and Truncation

`wrapped` (page 121) *property*
Determines whether the text is wrapped to fit within the receiver's bounds.

`alignmentMode` (page 118) *property*
Determines how individual lines of text are horizontally aligned within the receiver's bounds.

`truncationMode` (page 120) *property*
Determines how the text is truncated to fit within the receiver's bounds.

Properties

alignmentMode

Determines how individual lines of text are horizontally aligned within the receiver's bounds.

```
@property(copy) NSString *alignmentMode
```

Discussion
The possible values are described in `"Horizontal alignment modes"` (page 122). Defaults to `kCAAlignmentNatural` (page 122).

Availability
Available in iOS 3.2 and later.

Related Sample Code
AVSimpleEditoriOS

Declared in
`CATextLayer.h`

font

The font used to render the receiver's text.

```
@property CFTypeRef font
```

Discussion
May be either a `CTFontRef`, a `CGFontRef`, an instance of `NSFont` (OS X only), or a string naming the font. In iOS, you cannot assign a `UIFont` object to this property. Defaults to Helvetica.

The `font` property is only used when the `string` (page 120) property is not an `NSAttributedString`.

Note: If the font property is a `CTFontRef`, a `CGFontRef`, or an instance of `NSFont`, the font size of the property is ignored.

Availability
Available in iOS 3.2 and later.

Related Sample Code
AVSimpleEditoriOS

Declared in
`CATextLayer.h`

fontSize

The font size used to render the receiver's text. Animatable.

```
@property CGFloat fontSize
```

Discussion
Defaults to 36.0.

The `fontSize` property is only used when the `string` (page 120) property is not an `NSAttributedString`.

Note: Implicit animation of this property is only enabled in applications compiled for OS X v10.6 and later.

Availability
Available in iOS 3.2 and later.

Declared in
`CATextLayer.h`

foregroundColor

The color used to render the receiver's text. Animatable.

```
@property CGColorRef foregroundColor
```

Discussion
Defaults to opaque white.

The `foregroundColor` property is only used when the `string` (page 120) property is not an `NSAttributedString`.

Note: Implicit animation of this property is only enabled in applications compiled for OS X v10.6 and later.

Availability
Available in iOS 3.2 and later.

Related Sample Code
AVSimpleEditoriOS

Declared in
`CATextLayer.h`

string

The text to be rendered by the receiver.

```
@property(copy) id string
```

Discussion

The text must be an instance of `NSString` or `NSAttributedString`. Defaults to `nil`.

Availability

Available in iOS 3.2 and later.

Related Sample Code
AVSimpleEditoriOS

Declared in
`CATextLayer.h`

truncationMode

Determines how the text is truncated to fit within the receiver's bounds.

```
@property(copy) NSString *truncationMode
```

Discussion

The possible values are described in "`Truncation modes`" (page 121). Defaults to `kCATruncationNone` (page 121).

Availability

Available in iOS 3.2 and later.

Declared in
`CATextLayer.h`

wrapped

Determines whether the text is wrapped to fit within the receiver's bounds.

```
@property(getter=isWrapped) BOOL wrapped
```

Discussion
Defaults to `NO`.

Availability
Available in iOS 3.2 and later.

Declared in
`CATextLayer.h`

Constants

Truncation modes

These constants are used by the `truncationMode` (page 120) property.

```
NSString * const kCATruncationNone;
NSString * const kCATruncationStart;
NSString * const kCATruncationEnd;
NSString * const kCATruncationMiddle;
```

Constants
`kCATruncationNone`

If the `wrapped` (page 121) property is `YES`, the text is wrapped to the receiver's bounds, otherwise the text is clipped to the receiver's bounds.

Available in iOS 3.2 and later.

Declared in `CATextLayer.h`.

`kCATruncationStart`

Each line is displayed so that the end fits in the container and the missing text is indicated by some kind of ellipsis glyph.

Available in iOS 3.2 and later.

Declared in `CATextLayer.h`.

`kCATruncationEnd`

Each line is displayed so that the beginning fits in the container and the missing text is indicated by some kind of ellipsis glyph.

Available in iOS 3.2 and later.

Declared in `CATextLayer.h`.

`kCATruncationMiddle`

Each line is displayed so that the beginning and end fit in the container and the missing text is indicated by some kind of ellipsis glyph in the middle.

Available in iOS 3.2 and later.

Declared in `CATextLayer.h`.

Declared in
`CATextLayer.h`

Horizontal alignment modes

These constants are used by the `alignmentMode` *(page 118) property.*

```
NSString * const kCAAlignmentNatural;
NSString * const kCAAlignmentLeft;
NSString * const kCAAlignmentRight;
NSString * const kCAAlignmentCenter;
NSString * const kCAAlignmentJustified;
```

Constants

`kCAAlignmentNatural`

Use the natural alignment of the text's script.

Available in iOS 3.2 and later.

Declared in `CATextLayer.h`.

`kCAAlignmentLeft`

Text is visually left aligned.

Available in iOS 3.2 and later.

Declared in `CATextLayer.h`.

`kCAAlignmentRight`

Text is visually right aligned.

Available in iOS 3.2 and later.

Declared in `CATextLayer.h`.

`kCAAlignmentCenter`

Text is visually center aligned.

Available in iOS 3.2 and later.

Declared in `CATextLayer.h`.

`kCAAlignmentJustified`

Text is justified.

Available in iOS 3.2 and later.

Declared in `CATextLayer.h`.

Declared in
`CATextLayer.h`

CATiledLayer Class Reference

Inherits from	CALayer : NSObject
Conforms to	NSCoding (CALayer) CAMediaTiming (CALayer) NSObject (NSObject)
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in iOS 2.0 and later.
Declared in	CATiledLayer.h
Companion guides	Core Animation Programming Guide Core Animation Cookbook
Related sample code	PhotoScroller

Overview

`CATiledLayer` is a subclass of `CALayer` providing a way to asynchronously provide tiles of the layer's content, potentially cached at multiple levels of detail.

As more data is required by the renderer, the layer's `drawLayer:inContext:` method is called on one or more background threads to supply the drawing operations to fill in one tile of data. The clip bounds and CTM of the drawing context can be used to determine the bounds and resolution of the tile being requested.

Regions of the layer may be invalidated using the `setNeedsDisplayInRect:` (page 95) method however the update will be asynchronous. While the next display update will most likely not contain the updated content, a future update will.

Tasks

Visual Fade

`+ fadeDuration` (page 127)
  The time, in seconds, that newly added images take to "fade-in" to the rendered representation of the tiled layer.

Levels of Detail

`levelsOfDetail` (page 125) *property*
  The number of levels of detail maintained by this layer.

`levelsOfDetailBias` (page 126) *property*
  The number of magnified levels of detail for this layer.

Layer Tile Size

`tileSize` (page 126) *property*
  The maximum size of each tile used to create the layer's content.

Properties

levelsOfDetail

The number of levels of detail maintained by this layer.

```
@property size_t levelsOfDetail
```

Discussion
Defaults to 1. Each level of detail is half the resolution of the previous level. If too many levels are specified for the current size of the layer, then the number of levels is clamped to the maximum value (the bottom most level of detail must contain at least a single pixel in each dimension.)

Availability
Available in iOS 2.0 and later.

Related Sample Code
PhotoScroller

Declared in
`CATiledLayer.h`

levelsOfDetailBias

The number of magnified levels of detail for this layer.

```
@property size_t levelsOfDetailBias
```

Discussion
Defaults to 0. Each previous level of detail is twice the resolution of the later. For example, specifying a value of 2 means that the layer has two extra levels of detail: 2x and 4x.

Availability
Available in iOS 2.0 and later.

Declared in
`CATiledLayer.h`

tileSize

The maximum size of each tile used to create the layer's content.

```
@property CGSize tileSize
```

Discussion
Defaults to (256.0, 256.0).

Availability
Available in iOS 2.0 and later.

Related Sample Code
PhotoScroller

Declared in
`CATiledLayer.h`

Class Methods

fadeDuration

The time, in seconds, that newly added images take to "fade-in" to the rendered representation of the tiled layer.

```
+ (CFTimeInterval)fadeDuration
```

Discussion
The default implementation returns 0.25 seconds.

Availability
Available in iOS 2.0 and later.

Declared in
`CATiledLayer.h`

CATransaction Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in iOS 2.0 and later.
Declared in	CATransaction.h
Companion guides	Core Animation Programming Guide Core Animation Cookbook
Related sample code	AVLoupe oalTouch SquareCam UIKit Dynamics Catalog

Overview

`CATransaction` is the Core Animation mechanism for batching multiple layer-tree operations into atomic updates to the render tree. Every modification to a layer tree must be part of a transaction. Nested transactions are supported.

Core Animation supports two types of transactions: *implicit* transactions and *explicit* transactions. Implicit transactions are created automatically when the layer tree is modified by a thread without an active transaction and are committed automatically when the thread's run-loop next iterates. Explicit transactions occur when the the application sends the `CATransaction` class a `begin` (page 131) message before modifying the layer tree, and a `commit` (page 132) message afterwards.

`CATransaction` allows you to override default animation properties that are set for animatable properties. You can customize duration, timing function, whether changes to properties trigger animations, and provide a handler that informs you when all animations from the transaction group are completed.

During a transaction you can temporarily acquire a recursive spin-lock for managing property atomicity.

Tasks

Creating and Committing Transactions

`+ begin` (page 131)
Begin a new transaction for the current thread.

`+ commit` (page 132)
Commit all changes made during the current transaction.

`+ flush` (page 133)
Flushes any extant implicit transaction.

Overriding Animation Duration and Timing

`+ animationDuration` (page 130)
Returns the animation duration used by all animations within this transaction group.

`+ setAnimationDuration:` (page 134)
Sets the animation duration used by all animations within this transaction group.

`+ animationTimingFunction` (page 131)
Returns the timing function used for all animations within this transaction group.

`+ setAnimationTimingFunction:` (page 135)
Sets the timing function used for all animations within this transaction group.

Temporarily Disabling Property Animations

`+ disableActions` (page 133)
Returns whether actions triggered as a result of property changes made within this transaction group are suppressed.

`+ setDisableActions:` (page 136)
Sets whether actions triggered as a result of property changes made within this transaction group are suppressed.

Getting and Setting Completion Block Objects

`+ completionBlock` (page 132)
Returns the completion block object.

`+ setCompletionBlock:` (page 136)
  Sets the completion block object.

Managing Concurrency

`+ lock` (page 134)
  Attempts to acquire a recursive spin-lock lock, ensuring that returned layer values are valid until unlocked.

`+ unlock` (page 137)
  Relinquishes a previously acquired transaction lock.

Getting and Setting Transaction Properties

`+ setValue:forKey:` (page 137)
  Sets the arbitrary keyed-data for the specified key.

`+ valueForKey:` (page 138)
  Returns the arbitrary keyed-data specified by the given key.

Class Methods

animationDuration

Returns the animation duration used by all animations within this transaction group.

```
+ (CFTimeInterval)animationDuration
```

Return Value
An interval of time used as the duration.

Discussion
You can retrieve the animation duration for a specific transaction by calling the `valueForKey:` (page 138) method of the transaction object and asking for the `kCATransactionAnimationDuration` (page 139) key.

Availability
Available in iOS 3.0 and later.

See Also
`+ setAnimationDuration:` (page 134)

Declared in
`CATransaction.h`

animationTimingFunction

Returns the timing function used for all animations within this transaction group.

```
+ (CAMediaTimingFunction *)animationTimingFunction
```

Return Value
An instance of `CAMediaTimingFunction`.

Discussion
This is a convenience method that returns the `CAMediaTimingFunction` for the `valueForKey:` (page 138) value returned by the `kCATransactionAnimationTimingFunction` (page 139) key.

Availability
Available in iOS 3.0 and later.

See Also
`+ setAnimationTimingFunction:` (page 135)

Declared in
`CATransaction.h`

begin

Begin a new transaction for the current thread.

```
+ (void)begin
```

Discussion
The transaction is nested within the thread's current transaction, if there is one.

Availability
Available in iOS 2.0 and later.

See Also
`+ commit` (page 132)
`+ flush` (page 133)

Related Sample Code
AVLoupe
oalTouch

SquareCam

UIKit Dynamics Catalog

Declared in
`CATransaction.h`

commit

Commit all changes made during the current transaction.

```
+ (void)commit
```

Special Considerations
Raises an exception if no current transaction exists.

Availability
Available in iOS 2.0 and later.

See Also
`+ begin` (page 131)
`+ flush` (page 133)

Related Sample Code
AVLoupe

oalTouch

SquareCam

UIKit Dynamics Catalog

Declared in
`CATransaction.h`

completionBlock

Returns the completion block object.

```
+ (void)completionBlock
```

Discussion
See `setCompletionBlock:` (page 136) for a description of the role of the completion block object.

Availability
Available in iOS 4.0 and later.

See Also
`+ completionBlock` (page 132)

Declared in
`CATransaction.h`

disableActions

Returns whether actions triggered as a result of property changes made within this transaction group are suppressed.

```
+ (BOOL)disableActions
```

Return Value
`YES` if actions are disabled.

Discussion
This is a convenience method that returns the `boolValue` for the `valueForKey:` (page 138) value returned by the `kCATransactionDisableActions` (page 139) key.

Availability
Available in iOS 3.0 and later.

See Also
`+ setDisableActions:` (page 136)

Declared in
`CATransaction.h`

flush

Flushes any extant implicit transaction.

```
+ (void)flush
```

Discussion
Delays the commit until any nested explicit transactions have completed.

Flush is typically called automatically at the end of the current runloop, regardless of the runloop mode. If your application does not have a runloop, you must call this method explicitly.

However, you should attempt to avoid calling `flush` explicitly. By allowing `flush` to execute during the runloop your application will achieve better performance, atomic screen updates will be preserved, and transactions and animations that work from transaction to transaction will continue to function.

Availability
Available in iOS 2.0 and later.

See Also
`+ begin` (page 131)
`+ commit` (page 132)

Declared in
`CATransaction.h`

lock

Attempts to acquire a recursive spin-lock lock, ensuring that returned layer values are valid until unlocked.

```
+ (void)lock
```

Discussion
Core Animation uses a data model that promises not to corrupt the internal data structures when called from multiple threads concurrently, but not that data returned is still valid if the property was valid on another thread. By locking during a transaction you can ensure data that is read, modified, and set is correctly managed.

Availability
Available in iOS 3.0 and later.

See Also
`+ unlock` (page 137)

Declared in
`CATransaction.h`

setAnimationDuration:

Sets the animation duration used by all animations within this transaction group.

```
+ (void)setAnimationDuration:(CFTimeInterval)duration
```

Parameters

`duration`

An interval of time used as the duration.

Discussion

You can also set the animation duration for a specific transaction object by calling the `setValue:forKey:` (page 137) method of that object and specifying the `kCATransactionAnimationDuration` (page 139) key.

Availability

Available in iOS 3.0 and later.

See Also

`+ animationDuration` (page 130)

Related Sample Code

SquareCam

UIKit Dynamics Catalog

Declared in

`CATransaction.h`

setAnimationTimingFunction:

Sets the timing function used for all animations within this transaction group.

```
+ (void)setAnimationTimingFunction:(CAMediaTimingFunction *)function
```

Parameters

`function`

An instance of `CAMediaTimingFunction`.

Discussion

This is a convenience method that sets the `CAMediaTimingFunction` for the `valueForKey:` (page 138) value of the `kCATransactionAnimationTimingFunction` (page 139) key.

Availability

Available in iOS 3.0 and later.

See Also

`+ animationTimingFunction` (page 131)

Declared in

`CATransaction.h`

setCompletionBlock:

Sets the completion block object.

```
+ (void)setCompletionBlock:(void (^)(void))block
```

Parameters

`block`

A block object called when animations for this transaction group are completed.

The block object takes no parameters and returns no value.

Discussion

The completion block object that is guaranteed to be called (on the main thread) as soon as all animations subsequently added by this transaction group have completed (or have been removed.) If no animations are added before the current transaction group is committed (or the completion block is set to a different value,) the block will be invoked immediately.

Availability

Available in iOS 4.0 and later.

See Also

`+ completionBlock` (page 132)

Declared in

`CATransaction.h`

setDisableActions:

Sets whether actions triggered as a result of property changes made within this transaction group are suppressed.

```
+ (void)setDisableActions:(BOOL)flag
```

Parameters

`flag`

`YES`, if actions should be disabled.

Discussion

This is a convenience method that invokes `setValue:forKey:` (page 137) with an `NSNumber` containing a `YES` for the `kCATransactionDisableActions` (page 139) key.

Availability

Available in iOS 3.0 and later.

See Also

`+ disableActions` (page 133)

Related Sample Code
AVLoupe

Declared in

`CATransaction.h`

setValue:forKey:

Sets the arbitrary keyed-data for the specified key.

```
+ (void)setValue:(id)anObject forKey:(NSString *)key
```

Parameters

`anObject`
: The value for the key identified by `key`.

`key`
: The name of one of the receiver's properties.

Discussion

Nested transactions have nested data scope; setting a key always sets it in the innermost scope.

Availability

Available in iOS 2.0 and later.

Related Sample Code
oalTouch

SquareCam

Declared in

`CATransaction.h`

unlock

Relinquishes a previously acquired transaction lock.

```
+ (void)unlock
```

Availability

Available in iOS 3.0 and later.

See Also
`+ lock` (page 134)

Declared in
`CATransaction.h`

valueForKey:

Returns the arbitrary keyed-data specified by the given key.

```
+ (id)valueForKey:(NSString *)key
```

Parameters
`key`

The name of one of the receiver's properties.

Return Value
The value for the data specified by the key.

Discussion
Nested transactions have nested data scope. Requesting a value for a key first searches the innermost scope, then the enclosing transactions.

Availability
Available in iOS 2.0 and later.

Declared in
`CATransaction.h`

Constants

Transaction properties

These constants define the property keys used by `valueForKey:` (page 138) and `setValue:forKey:` (page 137).

```
NSString * const kCATransactionAnimationDuration;
NSString * const kCATransactionDisableActions;
NSString * const kCATransactionAnimationTimingFunction;
NSString * const kCATransactionCompletionBlock;
```

Constants

`kCATransactionAnimationDuration`

Duration, in seconds, for animations triggered within the transaction group. The value for this key must be an instance of `NSNumber`.

Available in iOS 2.0 and later.

Declared in `CATransaction.h`.

`kCATransactionDisableActions`

If `YES`, implicit actions for property changes made within the transaction group are suppressed. The value for this key must be an instance of `NSNumber`.

Available in iOS 2.0 and later.

Declared in `CATransaction.h`.

`kCATransactionAnimationTimingFunction`

An instance of `CAMediaTimingFunction` that overrides the timing function for all animations triggered within the transaction group.

Available in iOS 3.0 and later.

Declared in `CATransaction.h`.

`kCATransactionCompletionBlock`

A completion block object that is guaranteed to be called (on the main thread) as soon as all animations subsequently added by this transaction group have completed (or have been removed.) If no animations are added before the current transaction group is committed (or the completion block is set to a different value,) the block will be invoked immediately.

Available in iOS 4.0 and later.

Declared in `CATransaction.h`.

Declared in

`CATransaction.h`

CATransition Class Reference

Inherits from	CAAnimation : NSObject
Conforms to	NSCoding (CAAnimation) NSCopying (CAAnimation) CAAction (CAAnimation) CAMediaTiming (CAAnimation) NSObject (NSObject)
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in iOS 2.0 and later.
Declared in	CAAnimation.h
Companion guides	Core Animation Programming Guide Core Animation Cookbook

Overview

The `CATransition` class implements transition animations for a layer. You can specify the transition effect from a set of predefined transitions or by providing a custom `CIFilter` instance.

Tasks

Transition Start and End Point

`startProgress` (page 142) *property*
Indicates the start point of the receiver as a fraction of the entire transition.

`endProgress` (page 141) *property*
Indicates the end point of the receiver as a fraction of the entire transition.

Transition Properties

`type` (page 143) *property*
  Specifies the predefined transition type.

`subtype` (page 142) *property*
  Specifies an optional subtype that indicates the direction for the predefined motion-based transitions.

Custom Transition Filter

`filter` (page 141) *property*
  An optional Core Image filter object that provides the transition.

Properties

endProgress

Indicates the end point of the receiver as a fraction of the entire transition.

```
@property float endProgress
```

Discussion
The value must be greater than or equal to `startProgress` (page 142), and not greater than 1.0. If `endProgress` is less than `startProgress` (page 142) the behavior is undefined. The default value is 1.0.

Availability
Available in iOS 2.0 and later.

Declared in
`CAAnimation.h`

filter

An optional Core Image filter object that provides the transition.

```
@property(retain) CIFilter *filter
```

Discussion
If specified, the filter must support both `kCIInputImageKey` (page 188) and `kCIInputTargetImageKey` (page 190) input keys, and the `kCIOutputImageKey` (page 188) output key. The filter may optionally support the `kCIInputExtentKey` (page 190) input key, which is set to a rectangle describing the region in which the transition should run. If `filter` does not support the required input and output keys the behavior is undefined.

Defaults to `nil`. When a transition filter is specified the `type` (page 143) and `subtype` (page 142) properties are ignored.

Special Considerations
Core Image is available only in iOS 5 and later. In earlier versions of iOS, the filters you would assign to this property are unavailable and therefore cannot be used to create transitions.

Availability
Available in iOS 2.0 and later.

Declared in
`CAAnimation.h`

startProgress

Indicates the start point of the receiver as a fraction of the entire transition.

```
@property float startProgress
```

Discussion
Legal values are numbers between 0.0 and 1.0. For example, to start the transition half way through its progress set `startProgress` to 0.5. The default value is 0.

Availability
Available in iOS 2.0 and later.

Declared in
`CAAnimation.h`

subtype

Specifies an optional subtype that indicates the direction for the predefined motion-based transitions.

```
@property(copy) NSString *subtype
```

Discussion

The possible values are shown in "`Common Transition Subtypes`" (page 144). The default is `nil`.

This property is ignored if a custom transition is specified in the `filter` (page 141) property.

Availability

Available in iOS 2.0 and later.

Declared in

`CAAnimation.h`

type

Specifies the predefined transition type.

```
@property(copy) NSString *type
```

Discussion

The possible values are shown in "`Common Transition Types`" (page 143). This property is ignored if a custom transition is specified in the `filter` (page 141) property. The default is `kCATransitionFade` (page 144).

Availability

Available in iOS 2.0 and later.

Declared in

`CAAnimation.h`

Constants

Common Transition Types

These constants specify the transition types that can be used with the `type` (page 143) property.

```
NSString * const kCATransitionFade;
NSString * const kCATransitionMoveIn;
NSString * const kCATransitionPush;
NSString * const kCATransitionReveal;
```

Constants

`kCATransitionFade`

The layer's content fades as it becomes visible or hidden.

Available in iOS 2.0 and later.

Declared in `CAAnimation.h`.

`kCATransitionMoveIn`

The layer's content slides into place over any existing content. The "`Common Transition Subtypes`" (page 144) are used with this transition.

Available in iOS 2.0 and later.

Declared in `CAAnimation.h`.

`kCATransitionPush`

The layer's content pushes any existing content as it slides into place. The "`Common Transition Subtypes`" (page 144) are used with this transition.

Available in iOS 2.0 and later.

Declared in `CAAnimation.h`.

`kCATransitionReveal`

The layer's content is revealed gradually in the direction specified by the transition subtype. The "`Common Transition Subtypes`" (page 144) are used with this transition.

Available in iOS 2.0 and later.

Declared in `CAAnimation.h`.

Declared in

`CATransition.h`

Common Transition Subtypes

These constants specify the direction of motion-based transitions. They are used with the `subtype` (page 142) property.

```
NSString * const kCATransitionFromRight;
NSString * const kCATransitionFromLeft;
NSString * const kCATransitionFromTop;
NSString * const kCATransitionFromBottom;
```

Constants

`kCATransitionFromRight`

The transition begins at the right side of the layer.

Available in iOS 2.0 and later.

Declared in `CAAnimation.h`.

`kCATransitionFromLeft`

The transition begins at the left side of the layer.

Available in iOS 2.0 and later.

Declared in `CAAnimation.h`.

`kCATransitionFromTop`

The transition begins at the top of the layer.

Available in iOS 2.0 and later.

Declared in `CAAnimation.h`.

`kCATransitionFromBottom`

The transition begins at the bottom of the layer.

Available in iOS 2.0 and later.

Declared in `CAAnimation.h`.

Declared in

`CATransition.h`

CIColor Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	Library/Frameworks/CoreImage.framework
Availability	Available in iOS 5.0 and later.
Declared in	CIColor.h
Companion guides	Core Image Programming Guide Color Management Overview
Related sample code	Core Image Filters with Photos and Video for iOS

Overview

The `CIColor` class contains color values and the color space for which the color values are valid. You use `CIColor` objects in conjunction with other Core Image classes, such as `CIFilter`, `CIContext`,and `CIImage`, to take advantage of the built-in Core Image filters when processing images.

A color space defines a one-, two-, three-, or four-dimensional environment whose color components represent intensity values. A color component is also referred to as a color channel. An RGB color space, for example, is a three-dimensional color space whose stimuli are the red, green, and blue intensities that make up a given color. Regardless of the color space, in Core Image, color values range from `0.0` to `1.0`, with `0.0` representing an absence of that component (0 percent) and `1.0` representing 100 percent.

Colors also have an alpha component that represents the opacity of the color, with `0.0` meaning completely transparent and `1.0` meaning completely opaque. If a color does not have an explicit alpha component, Core Image paints the color as if the alpha component equals `1.0`. You always provide unpremultiplied color components to Core Image and Core Image provides unpremultiplied color components to you. Core Image premultiplies each color component with the alpha value in order to optimize calculations. For more information on premultiplied alpha values see *Core Image Programming Guide* .

Tasks

Initializing Color Objects

– `initWithCGColor:` (page 153)
  Initializes a color object with a Quartz color.

Creating Color Objects

+ `colorWithCGColor:` (page 148)
  Creates a color object from a Quartz color.

+ `colorWithRed:green:blue:` (page 149)
  Creates a color object using the specified RGB color component values

+ `colorWithRed:green:blue:alpha:` (page 149)
  Creates a color object using the specified RGBA color component values.

+ `colorWithString:` (page 150)
  Creates a color object using the RGBA color component values specified by a string.

Getting Color Components

– `alpha` (page 151)
  Returns the alpha value of the color.

– `blue` (page 151)
  Returns the blue component of the color.

– `colorSpace` (page 152)
  Returns the Quartz 2D color space associated with the color.

– `components` (page 152)
  Returns the color components of the color.

– `green` (page 153)
  Returns the green component of the color.

– `numberOfComponents` (page 154)
  Returns the number of color components in the color.

– `red` (page 154)
  Returns the red component of the color.

– `stringRepresentation` (page 154)
Returns a formatted string that specifies the components of the color.

Class Methods

colorWithCGColor:

Creates a color object from a Quartz color.

```
+ (CIColor *)colorWithCGColor:(CGColorRef)c
```

Parameters

`c`
A Quartz color (`CGColorRef` object) created using a Quartz color creation function such as `CGColorCreate`.

Return Value
A Core Image color object that represents a Quartz color.

Discussion
A `CGColorRef` object is the fundamental opaque data type used internally by Quartz to represent colors. For more information on Quartz 2D color and color spaces, see *Quartz 2D Programming Guide* .

You can pass a `CGColorRef` object that represents any color space, including CMYK, but Core Image converts all color spaces to the Core Image working color space before it passes the color space to the filter kernel. The Core Image working color space uses three color components plus alpha.

Availability
Available in iOS 5.0 and later.

See Also
+ `colorWithRed:green:blue:` (page 149)
+ `colorWithRed:green:blue:alpha:` (page 149)
+ `colorWithString:` (page 150)

Declared in
`CIColor.h`

colorWithRed:green:blue:

Creates a color object using the specified RGB color component values

```
+ (CIColor *)colorWithRed:(CGFloat)r green:(CGFloat)g blue:(CGFloat)b
```

Parameters

`r`

The value of the red component.

`g`

The value of the green component.

`b`

The value of the blue component.

Return Value

A Core Image color object that represents an RGB color in the color space specified by the Quartz 2D constant `kCGColorSpaceGenericRGB`.

Availability

Available in iOS 5.0 and later.

See Also

`+ colorWithCGColor:` (page 148)
`+ colorWithRed:green:blue:alpha:` (page 149)
`+ colorWithString:` (page 150)

Declared in

`CIColor.h`

colorWithRed:green:blue:alpha:

Creates a color object using the specified RGBA color component values.

```
+ (CIColor *)colorWithRed:(CGFloat)r green:(CGFloat)g blue:(CGFloat)b alpha:(CGFloat)a
```

Parameters

`r`

The value of the red component.

`g`

The value of the green component.

`b`

The value of the blue component.

`a`

The value of the alpha component.

Return Value

A Core Image color object that represents an RGB color in the color space specified by the Quartz 2D constant `kCGColorSpaceGenericRGB` and an alpha value.

Availability

Available in iOS 5.0 and later.

See Also

`+ colorWithCGColor:` (page 148)
`+ colorWithRed:green:blue:` (page 149)
`+ colorWithString:` (page 150)

Related Sample Code

Core Image Filters with Photos and Video for iOS

Declared in

`CIColor.h`

colorWithString:

Creates a color object using the RGBA color component values specified by a string.

```
+ (CIColor *)colorWithString:(NSString *)representation
```

Parameters

`representation`

A string that is in one of the formats returned by the `stringRepresentation` method. For example, the string:

```
@"0.5 0.7 0.3 1.0"
```

indicates an RGB color whose components are 50% red, 70% green, 30% blue, and 100% opaque (alpha value of 1.0). The string representation always has four components—red, green, blue, and alpha. The default value for the alpha component is `1.0`.

Return Value

A Core Image color object that represents an RGB color in the color space specified by the Quartz 2D constant `kCGColorSpaceGenericRGB`.

Availability

Available in iOS 5.0 and later.

See Also

`+ colorWithCGColor:` (page 148)

`+ colorWithRed:green:blue:` (page 149)

`+ colorWithRed:green:blue:alpha:` (page 149)

Declared in

`CIColor.h`

Instance Methods

alpha

Returns the alpha value of the color.

```
- (CGFloat)alpha
```

Return Value

The alpha value. A color created without an explicit alpha value has an alpha of 1.0 by default.

Availability

Available in iOS 5.0 and later.

See Also

`– components` (page 152)

Declared in

`CIColor.h`

blue

Returns the blue component of the color.

```
- (CGFloat)blue
```

Return Value

The unpremultiplied blue component of the color.

Availability

Available in iOS 5.0 and later.

See Also

`– components` (page 152)

Declared in
`CIColor.h`

colorSpace

Returns the Quartz 2D color space associated with the color.

```
- (CGColorSpaceRef)colorSpace
```

Return Value
The Quartz 2D color space (`CGColorSpaceRef` object).

Availability
Available in iOS 5.0 and later.

See Also
– `components` (page 152)

Declared in
`CIColor.h`

components

Returns the color components of the color.

```
- (const CGFloat *)components
```

Return Value
An array of color components, specified as floating-point values in the range of 0.0 through 1.0. This array includes an alpha component if there is one.

Availability
Available in iOS 5.0 and later.

See Also
– `numberOfComponents` (page 154)
– `stringRepresentation` (page 154)

Declared in
`CIColor.h`

green

Returns the green component of the color.

```
- (CGFloat)green
```

Return Value

The unpremultiplied green component of the color.

Availability

Available in iOS 5.0 and later.

See Also

– `components` (page 152)

Declared in

`CIColor.h`

initWithCGColor:

Initializes a color object with a Quartz color.

```
- (id)initWithCGColor:(CGColorRef)c
```

Parameters

`c`

A Quartz color (`CGColorRef`) created using a Quartz color creation function such as `CGColorCreate`.

Discussion

A `CGColorRef` object is the fundamental opaque data type used internally by Quartz to represent colors. For more information on Quartz 2D color and color spaces, see *Quartz 2D Programming Guide* .

You can pass a `CGColorRef` object that represents any color space, including CMYK, but Core Image converts all color spaces to the Core Image working color space before it passes the color space to the filter kernel. The Core Image working color space uses three color components plus alpha.

Availability

Available in iOS 5.0 and later.

Declared in

`CIColor.h`

numberOfComponents

Returns the number of color components in the color.

```
- (size_t)numberOfComponents
```

Return Value
The number of color components, which includes an alpha component if there is one.

Availability
Available in iOS 5.0 and later.

See Also
– `components` (page 152)

Declared in
`CIColor.h`

red

Returns the red component of the color.

```
- (CGFloat)red
```

Return Value
The unpremultiplied red component of the color.

Availability
Available in iOS 5.0 and later.

See Also
– `components` (page 152)

Declared in
`CIColor.h`

stringRepresentation

Returns a formatted string that specifies the components of the color.

```
- (NSString *)stringRepresentation
```

Return Value

The formatted string.

Discussion

The string representation always has four components—red, green, blue, and alpha. The default value for the alpha component is `1.0`.F or example, this string:

```
@"0.5 0.7 0.3 1.0"
```

indicates an RGB color whose components are 50% red, 70% green, 30% blue, and 100% opaque (alpha value of 1.0).

Availability

Available in iOS 5.0 and later.

See Also

– `components` (page 152)

Declared in

`CIColor.h`

CIContext Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	Library/Frameworks/CoreImage.framework
Availability	Available in iOS 5.0 and later.
Declared in	CIContext.h
Companion guides	Core Image Programming Guide Image Unit Tutorial
Related sample code	AirDrop Examples Core Image Filters with Photos and Video for iOS

Overview

The `CIContext` class provides an evaluation context for rendering a `CIImage` object through Quartz 2D or OpenGL. You use `CIContext` objects in conjunction with other Core Image classes, such as `CIFilter`, `CIImage`, and `CIColor`, to take advantage of the built-in Core Image filters when processing images.

`CIContext` and `CIImage` objects are immutable, which means each can be shared safely among threads. Multiple threads can use the same GPU or CPU `CIContext` object to render `CIImage` objects. However, this is not the case for `CIFilter` objects, which are mutable. A `CIFilter` object cannot be shared safely among threads. If you app is multithreaded, each thread must create its own `CIFilter` objects. Otherwise, your app could behave unexpectedly.

Tasks

Creating a Context

+ `contextWithEAGLContext:` (page 158)
  Creates a Core Image context from an EAGL context.

+ `contextWithEAGLContext:options:` (page 158)
  Creates a Core Image context from an EAGL context using the specified options.

+ `contextWithOptions:` (page 159)
  Creates a CPU-based Core Image context using the specified options.

Rendering Images

– `createCGImage:fromRect:` (page 160)
  Creates a Quartz 2D image from a region of a Core Image image object.

– `createCGImage:fromRect:format:colorSpace:` (page 160)
  Creates a Quartz 2D image from a region of a Core Image image object.

– `drawImage:atPoint:fromRect:` (page 161)
  Renders a region of an image to a point in the context destination. (Deprecated. Instead use `drawImage:inRect:fromRect:` (page 162).)

– `drawImage:inRect:fromRect:` (page 162)
  Renders a region of an image to a rectangle in the context destination.

– `render:toBitmap:rowBytes:bounds:format:colorSpace:` (page 164)
  Renders to the given bitmap.

– `render:toCVPixelBuffer:` (page 164)
  Renders an image into a pixel buffer.

– `render:toCVPixelBuffer:bounds:colorSpace:` (page 165)
  Renders a region of an image into a pixel buffer.

Determining the Allowed Extents for Images Used by a Context

– `inputImageMaximumSize` (page 163)
  Returns the maximum size allowed for any image rendered into the context.

– `outputImageMaximumSize` (page 163)
  Returns the maximum size allowed for any image created by the context.

Class Methods

contextWithEAGLContext:

Creates a Core Image context from an EAGL context.

```
+ (CIContext *)contextWithEAGLContext:(EAGLContext *)eaglContext
```

Parameters

`eaglContext`

The EAGL context to render to.

Return Value

A Core Image context that targets OpenGL ES.

Discussion

The OpenGL ES context must support OpenGL ES 2.0. All drawing performed using the `drawImage:atPoint:fromRect:` (page 161) method or the `drawImage:inRect:fromRect:` (page 162) method is rendered directly into the context.

Availability

Available in iOS 5.0 and later.

Declared in

`CIContext.h`

contextWithEAGLContext:options:

Creates a Core Image context from an EAGL context using the specified options.

```
+ (CIContext *)contextWithEAGLContext:(EAGLContext *)eaglContext options:(NSDictionary
 *)dict
```

Parameters

`eaglContext`

The EAGL context to render to.

`options`

A dictionary that contains options for creating a `CIContext` object. You can pass any of the keys defined in "`Context Options`" (page 165) along with the appropriate value.

Return Value

A Core Image context that targets OpenGL ES.

Discussion

The OpenGL ES context must support OpenGL ES 2.0. All drawing performed using the `drawImage:atPoint:fromRect:` (page 161) method or the `drawImage:inRect:fromRect:` (page 162) method is rendered directly into the context.

You should use this method if you want to get real-time performance on a device. One of the advantages of using an EAGL context is that the rendered image stays on the GPU and does not get copied to CPU memory.

Availability

Available in iOS 5.0 and later.

Related Sample Code
Core Image Filters with Photos and Video for iOS

Declared in

`CIContext.h`

contextWithOptions:

Creates a CPU-based Core Image context using the specified options.

```
+ (CIContext *)contextWithOptions:(NSDictionary *)dict
```

Parameters

`dict`

A dictionary that contains options for the context. You can pass any of the keys defined in "`Context Options`" (page 165) along with the appropriate value.

Return Value

A Core Image context.

Discussion

You can create a CPU-based context by providing the key `kCIContextUseSoftwareRenderer` (page 166). A CPU-based context supports larger input and output images than a GPU-based context. It also allows your app to perform processing in the background, such as when saving the rendered output to the Photo Library.

GPU rendering is faster than CPU rendering, but the resulting image is not displayed on the device until after is it copied to CPU memory and converted to another image type, such as a `UIImage` object.

Availability

Available in iOS 5.0 and later.

See Also

`+ contextWithEAGLContext:` (page 158)

`+ contextWithEAGLContext:options:` (page 158)

Related Sample Code
AirDrop Examples

Declared in
`CIContext.h`

Instance Methods

createCGImage:fromRect:

Creates a Quartz 2D image from a region of a Core Image image object.

```
- (CGImageRef)createCGImage:(CIImage *)im fromRect:(CGRect)r
```

Parameters

`im`
A Core Image image object.

`r`
The region of the image to render.

Return Value
A Quartz 2D image. You are responsible for releasing the returned image when you no longer need it.

Discussion
Renders a region of an image into a temporary buffer using the context, then creates and returns a Quartz 2D image with the results.

Availability
Available in iOS 5.0 and later.

See Also
`createCGImage:fromRect:format:colorSpace:` (page 160)

Declared in
`CIContext.h`

createCGImage:fromRect:format:colorSpace:

Creates a Quartz 2D image from a region of a Core Image image object.

```
- (CGImageRef)createCGImage:(CIImage *)im fromRect:(CGRect)r format:(CIFormat)f
colorSpace:(CGColorSpaceRef)cs
```

Parameters

`im`

A Core Image image object.

`r`

The region of the image to render.

`f`

The format of the image.

`cs`

The color space of the image.

Return Value

A Quartz 2D image. You are responsible for releasing the returned image when you no longer need it.

Discussion

Renders a region of an image into a temporary buffer using the context, then creates and returns a Quartz 2D image with the results.

Availability

Available in iOS 5.0 and later.

See Also

`createCGImage:fromRect:` (page 160)

Declared in

`CIContext.h`

drawImage:atPoint:fromRect:

Renders a region of an image to a point in the context destination. (Deprecated in iOS 6.0. Instead use `drawImage:inRect:fromRect:` (page 162).)

```
- (void)drawImage:(CIImage *)im atPoint:(CGPoint)p fromRect:(CGRect)src
```

Parameters

`im`

A Core Image image object.

`p`

The point in the context destination to draw to.

`src`
: The region of the image to draw.

Discussion

This method because it is ambiguous as to the units of the dimensions and won't work as expected in a high-resolution environment which is why you should use `drawImage:inRect:fromRect:` instead.

On iOS platforms, this method draws the image onto a render buffer for the OpenGL ES context. Use this method only if the `CIContext` object is created with `contextWithEAGLContext:`, and hence, you are rendering to a CAEAGLLayer.

Availability

Available in iOS 5.0 and later.

Deprecated in iOS 6.0.

See Also

– `drawImage:inRect:fromRect:` (page 162)

Declared in

`CIContext.h`

drawImage:inRect:fromRect:

Renders a region of an image to a rectangle in the context destination.

```
- (void)drawImage:(CIImage *)im inRect:(CGRect)dest fromRect:(CGRect)src
```

Parameters

`im`
: A `CIImage` object.

`dest`
: The rectangle in the context destination to draw into. The image is scaled to fill the destination rectangle.

`src`
: The subregion of the image that you want to draw into the context, with the origin and target size defined by the `dest` parameter. This rectangle is always in pixel dimensions.

Discussion

On iOS, this method draws the `CIImage` object into a renderbuffer for the OpenGL ES context. Use this method only if the `CIContext` object is created with `contextWithEAGLContext:` and if you are rendering to a CAEAGLayer.

On OS X, you need to be aware of whether the `CIContext` object is created with a `CGContextRef` or a `CGLContext` object. If you create the `CIContext` object with a `CGContextRef`, the dimensions of the destination rectangle are in points. If you create the `CIContext` object with a `CGLContext` object, the dimensions are in pixels.

On iOS 5, this method is synchronous. On iOS 6, this method is asynchronous. For apps linked on iOS 5, this method will continue to be synchronous.

Availability
Available in iOS 5.0 and later.

Declared in
`CIContext.h`

inputImageMaximumSize

Returns the maximum size allowed for any image rendered into the context.

```
- (CGSize)inputImageMaximumSize
```

Discussion
Some contexts limit the maximum size of an image that can be rendered into them. For example, the maximum size might reflect a limitation in the underlying graphics hardware.

Availability
Available in iOS 5.0 and later.

Declared in
`CIContext.h`

outputImageMaximumSize

Returns the maximum size allowed for any image created by the context.

```
- (CGSize)outputImageMaximumSize
```

Discussion
Some contexts limit the maximum size of an image that can be created by them. For example, the maximum size might reflect a limitation in the underlying graphics hardware.

Availability
Available in iOS 5.0 and later.

Declared in
`CIContext.h`

render:toBitmap:rowBytes:bounds:format:colorSpace:

Renders to the given bitmap.

```
- (void)render:(CIImage *)im toBitmap:(void *)data rowBytes:(ptrdiff_t)rb
bounds:(CGRect)r format:(CIFormat)f colorSpace:(CGColorSpaceRef)cs
```

Parameters

`im`
    A Core Image image object.

`data`
    Storage for the bitmap data.

`rb`
    The bytes per row.

`r`
    The bounds of the bitmap data.

`f`
    The format of the bitmap data.

`cs`
    The color space for the data. Pass `NULL` if you want to use the output color space of the context.

Availability
Available in iOS 5.0 and later.

Declared in
`CIContext.h`

render:toCVPixelBuffer:

Renders an image into a pixel buffer.

```
- (void)render:(CIImage *)image toCVPixelBuffer:(CVPixelBufferRef)buffer
```

Parameters

`image`
    A Core Image image object.

`buffer`
The destination pixel buffer.

Availability
Available in iOS 5.0 and later.

Declared in
`CIContext.h`

render:toCVPixelBuffer:bounds:colorSpace:

Renders a region of an image into a pixel buffer.

```
- (void)render:(CIImage *)image toCVPixelBuffer:(CVPixelBufferRef)buffer
bounds:(CGRect)r colorSpace:(CGColorSpaceRef)cs
```

Parameters

`image`
A Core Image image object.

`buffer`
The destination pixel buffer.

`r`
The rectangle in the destination pixel buffer to draw into.

`cs`
The color space of the destination pixel buffer.

Availability
Available in iOS 5.0 and later.

Declared in
`CIContext.h`

Constants

Context Options

Keys in the options dictionary for a `CIContext` object.

```
extern NSString *kCIContextOutputColorSpace;
extern NSString *kCIContextWorkingColorSpace;
extern NSString *kCIContextUseSoftwareRenderer;
```

Constants

`kCIContextOutputColorSpace`

A key for the color space to use for images before they are rendered to the context. By default, Core Image uses the GenericRGB color space, which leaves color matching to the system. You can specify a different output color space by providing a Quartz 2D CGColorSpace object (`CGColorSpaceRef`). (See *Quartz 2D Programming Guide* for information on creating and using CGColorSpace objects.)

Available in iOS 5.0 and later.

Declared in `CIContext.h`.

`kCIContextWorkingColorSpace`

A key for the color space to use for image operations. By default, Core Image assumes that processing nodes are 128 bits-per-pixel, linear light, premultiplied RGBA floating-point values that use the GenericRGB color space. You can specify a different working color space by providing a Quartz 2D CGColorSpace object (`CGColorSpaceRef`). Note that the working color space must be RGB-based. If you have YUV data as input (or other data that is not RGB-based), you can use ColorSync functions to convert to the working color space. (See *Quartz 2D Programming Guide* for information on creating and using CGColorSpace objects.)

Available in iOS 5.0 and later.

Declared in `CIContext.h`.

`kCIContextUseSoftwareRenderer`

A key for enabling software renderer use. If the associated `NSNumber` object is `YES`, then the software renderer is required.

Available in iOS 5.0 and later.

Declared in `CIContext.h`.

Discussion

For a discussion of when to use options and color management, see *Core Image Programming Guide*.

Declared in

`CIContext.h`

CIFilter Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	Library/Frameworks/CoreImage.framework
Availability	Available in iOS 5.0 and later.
Declared in	CIFilter.h CIRAWFilter.h
Companion guides	Core Image Programming Guide Image Unit Tutorial Core Image Filter Reference
Related sample code	AirDrop Examples Core Image Filters with Photos and Video for iOS PocketCoreImage

Overview

The `CIFilter` class produces a `CIImage` object as output. Typically, a filter takes one or more images as input. Some filters, however, generate an image based on other types of input parameters. The parameters of a `CIFilter` object are set and retrieved through the use of key-value pairs.

You use the `CIFilter` object in conjunction with other Core Image classes, such as `CIImage`, `CIContext`, and `CIColor`, to take advantage of the built-in Core Image filters when processing images, creating filter generators, or writing custom filters.

`CIFilter` objects, are not mutable. A `CIFilter` object cannot be shared safely among threads. If you app is multithreaded, each thread must create its own `CIFilter` object. Otherwise, your app could behave unexpectedly.

To get a quick overview of how to set up and use Core Image filters, see *Core Image Programming Guide*.

Subclassing Notes

You can subclass `CIFilter` in order to create custom filter effects:

- By chaining together two or more built-in Core Image filters (iOS and OS X)
- By using an image-processing kernel that you write (OS X only)

See *Core Image Programming Guide* for details.

Methods to Override

Regardless of whether your subclass provides its effect by chaining filters or implementing its own kernel, you should:

- Declare any input parameters as properties whose names are prefixed with `input`, such as `inputImage`.
- Override the `setDefaults` (page 178) methods to provide default values for any input parameters you've declared.
- Implement an `outputImage` method to create a new `CIImage` with your filter's effect.

Special Considerations

The `CIFilter` class automatically manages input parameters when archiving, copying, and deallocating filters. For this reason, your subclass must obey the following guidelines to ensure proper behavior:

- Store input parameters in instance variables whose names are prefixed with `input`.

  Don't use auto-synthesized instance variables, because their names are automatically prefixed with an underscore. Instead, synthesize the property manually. For example:

  ```
  @synthesize inputMyParameter;
  ```
- If using manual reference counting, don't release input parameter instance variables in your `dealloc` method implementation. The `dealloc` implementation in the `CIFilter` class uses key-value coding to automatically sets the values of all input parameters to `nil`.

Tasks

Creating a Filter

`+ filterWithName:` (page 173)
  Creates a `CIFilter` object for a specific kind of filter.

`+ filterWithName:keysAndValues:` (page 173)
  Creates a `CIFilter` object for a specific kind of filter and initializes the input values.

Accessing Registered Filters

`+ filterNamesInCategories:` (page 171)
  Returns an array of all published filter names that match all the specified categories.

`+ filterNamesInCategory:` (page 172)
  Returns an array of all published filter names in the specified category.

Getting Filter Parameters and Attributes

`– attributes` (page 175)
  Returns a dictionary of key-value pairs that describe the filter.

`– inputKeys` (page 177)
  Returns an array that contains the names of the input parameters to the filter.

`– outputKeys` (page 177)
  Returns an array that contains the names of the output parameters for the filter.

`– name` (page 177)
  The name of the filter.

`outputImage` (page 170) *property*
  Returns a `CIImage` object that encapsulates the operations configured in the filter. (read-only)

Setting Default Values

`– setDefaults` (page 178)
  Sets all input values for a filter to default values.

Using Filters with Core Animation

`name` (page 170) *property*
  A name associated with a filter.

Serializing and Deserializing Filters

`+ serializedXMPFromFilters:inputImageExtent:` (page 174)
  Serializes filter parameters into XMP form that is suitable for embedding in an image.

`+ filterArrayFromSerializedXMP:inputImageExtent:error:` (page 171)
  Returns an array of filter objects de-serialized from XMP data.

Properties

name

A name associated with a filter.

```
@property(copy) NSString *name
```

Discussion
You use a filter's name to construct key paths to its attributes when the filter is attached to a Core Animation layer. For example, if a `CALayer` object has an attached `CIFilter` instance whose name is `myExposureFilter`, you can refer to attributes of the filter using a key path such as `filters.myExposureFilter.inputEV`. Layer animations may also access filter attributes via these key paths.

The default value for this property is `nil`.

Availability
Available in iOS 5.0 and later.

Declared in
`CIFilter.h`

outputImage

Returns a `CIImage` object that encapsulates the operations configured in the filter. (read-only)

```
@property(readonly, nonatomic) CIImage *outputImage
```

Availability
Available in iOS 5.0 and later.

Related Sample Code
Core Image Filters with Photos and Video for iOS

Declared in
`CIFilter.h`

Class Methods

filterArrayFromSerializedXMP:inputImageExtent:error:

Returns an array of filter objects de-serialized from XMP data.

```
+ (NSArray*)filterArrayFromSerializedXMP:(NSData *)xmpData
inputImageExtent:(CGRect)extent error:(NSError **)outError
```

Parameters

`xmpData`
The XMP data created previously by calling `serializedXMPFromFilters:inputImageExtent:` (page 174).

`extent`
The extent of the image from which the XMP data was extracted.

`outError`
The address of an `NSError` object for receiving errors, otherwise `nil`.

Availability
Available in iOS 6.0 and later.

Declared in
`CIFilter.h`

filterNamesInCategories:

Returns an array of all published filter names that match all the specified categories.

```
+ (NSArray *)filterNamesInCategories:(NSArray *)categories
```

Parameters

`categories`

One or more of the filter category keys defined in "Filter Category Keys" (page 184). Pass `nil` to get all filters in all categories.

Return Value

An array that contains all published filter names that match all the categories specified by the `categories` argument.

Discussion

When you pass more than one filter category, this method returns the intersection of the filters in the categories. For example, if you pass the categories `kCICategoryBuiltIn` (page 187) and `kCICategoryColorAdjustment` (page 185), you obtain all the filters that are members of both the built-in and color adjustment categories. But if you pass in `kCICategoryGenerator` and `kCICategoryStylize` (page 186), you will not get any filters returned to you because there are no filters that are members of both the generator and stylize categories. If you want to obtain all stylize and generator filters, you must call the `filterNamesInCategories:` method for each category separately and then merge the results.

Availability

Available in iOS 5.0 and later.

See Also

`+ filterNamesInCategory:` (page 172)

Declared in

`CIFilter.h`

filterNamesInCategory:

Returns an array of all published filter names in the specified category.

```
+ (NSArray *)filterNamesInCategory:(NSString *)category
```

Parameters

`category`

A string object that specifies one of the filter categories defined in "Filter Category Keys" (page 184).

Return Value

An array that contains all published names of the filter in a category.

Availability

Available in iOS 5.0 and later.

See Also
`+ filterNamesInCategories:` (page 171)

Declared in
`CIFilter.h`

filterWithName:

Creates a `CIFilter` object for a specific kind of filter.

```
+ (CIFilter *)filterWithName:(NSString *)name
```

Parameters
`name`

The name of the filter. You must make sure the name is spelled correctly, otherwise your app will run but not produce any output images. For that reason, you should check for the existence of the filter after calling this method.

Return Value
A `CIFilter` object whose input values are undefined.

Discussion
You should call `setDefaults` (page 178) after you call this method or set values individually by calling `setValue:forKey`.

Availability
Available in iOS 5.0 and later.

See Also
`+ filterWithName:keysAndValues:` (page 173)

Related Sample Code
AirDrop Examples

Core Image Filters with Photos and Video for iOS

PocketCoreImage

Declared in
`CIFilter.h`

filterWithName:keysAndValues:

Creates a `CIFilter` object for a specific kind of filter and initializes the input values.

```
+ (CIFilter *)filterWithName:(NSString *)name keysAndValues:key0, ...
```

Parameters

`name`

The name of the filter. You must make sure the name is spelled correctly, otherwise your app will run but not produce any output images. For that reason, you should check for the existence of the filter after calling this method.

`key0,`

A list of key-value pairs to set as input values to the filter. Each key is a constant that specifies the name of the input value to set, and must be followed by a value. You signal the end of the list by passing a `nil` value.

Return Value

A `CIFilter` object whose input values are initialized.

Availability

Available in iOS 5.0 and later.

See Also

`+ filterWithName:` (page 173)

Related Sample Code
Core Image Filters with Photos and Video for iOS

Declared in

`CIFilter.h`

serializedXMPFromFilters:inputImageExtent:

Serializes filter parameters into XMP form that is suitable for embedding in an image.

```
+ (NSData*)serializedXMPFromFilters:(NSArray *)filters inputImageExtent:(CGRect)extent
```

Parameters

`filters`

The array of filters to serialize. See Discussion for the filters that can be serialized.

`extent`

The extent of the input image to the filter.

Discussion
At this time the only filters classes that can be serialized using this method are, CIAffineTransform, CICrop, and the filters returned by the `CIImage` methods `autoAdjustmentFilters` and `autoAdjustmentFiltersWithOptions:`. The parameters of other filter classes will not be serialized.

Availability
Available in iOS 6.0 and later.

Declared in
`CIFilter.h`

Instance Methods

attributes

Returns a dictionary of key-value pairs that describe the filter.

```
- (NSDictionary *)attributes
```

Return Value
A dictionary that contains a key for each input and output parameter for the filter. Each key is a dictionary that contains all the attributes of an input or output parameter.

Discussion
For example, the attributes dictionary for the `CIColorControls` filter contains the following information:

```
CIColorControls:
{
    CIAttributeFilterCategories = (
        CICategoryColorAdjustment,
        CICategoryVideo,
        CICategoryStillImage,
        CICategoryInterlaced,
        CICategoryNonSquarePixels,
        CICategoryBuiltIn
    );
    CIAttributeFilterDisplayName = "Color Controls";
    CIAttributeFilterName = CIColorControls;
    inputBrightness = {
        CIAttributeClass = NSNumber;
        CIAttributeDefault = 0;
        CIAttributeIdentity = 0;
        CIAttributeMin = -1;
        CIAttributeSliderMax = 1;
        CIAttributeSliderMin = -1;
        CIAttributeType = CIAttributeTypeScalar;
    };
    inputContrast = {
        CIAttributeClass = NSNumber;
        CIAttributeDefault = 1;
        CIAttributeIdentity = 1;
        CIAttributeMin = 0.25;
        CIAttributeSliderMax = 4;
        CIAttributeSliderMin = 0.25;
        CIAttributeType = CIAttributeTypeScalar;
    };
    inputImage = {CIAttributeClass = CIImage; };
    inputSaturation = {
        CIAttributeClass = NSNumber;
        CIAttributeDefault = 1;
        CIAttributeIdentity = 1;
        CIAttributeMin = 0;
        CIAttributeSliderMax = 3;
        CIAttributeSliderMin = 0;
        CIAttributeType = CIAttributeTypeScalar;
    };
    outputImage = {CIAttributeClass = CIImage; };
}
```

Availability
Available in iOS 5.0 and later.

Related Sample Code
Core Image Filters with Photos and Video for iOS

PocketCoreImage

Declared in
`CIFilter.h`

inputKeys

Returns an array that contains the names of the input parameters to the filter.

```
- (NSArray *)inputKeys
```

Return Value
An array that contains the names of all input parameters to the filter.

Availability
Available in iOS 5.0 and later.

Related Sample Code
Core Image Filters with Photos and Video for iOS

Declared in
`CIFilter.h`

name

The name of the filter.

```
- (NSString *)name
```

Return Value
A string that holds the name of the filter.

Availability
Available in iOS 5.0 and later.

Related Sample Code
Core Image Filters with Photos and Video for iOS

Declared in
`CIFilter.h`

outputKeys

Returns an array that contains the names of the output parameters for the filter.

```
- (NSArray *)outputKeys
```

Return Value

An array that contains the names of all output parameters from the filter.

Availability

Available in iOS 5.0 and later.

Declared in

`CIFilter.h`

setDefaults

Sets all input values for a filter to default values.

```
- (void)setDefaults
```

Discussion

Input values whose default values are not defined are left unchanged.

Availability

Available in iOS 5.0 and later.

Related Sample Code

Core Image Filters with Photos and Video for iOS

PocketCoreImage

Declared in

`CIFilter.h`

Constants

Filter Attribute Keys

Attributes for a filter and its parameters.

```
extern NSString *kCIAttributeFilterName;
extern NSString *kCIAttributeFilterDisplayName;
extern NSString *kCIAttributeFilterCategories;
extern NSString *kCIAttributeClass;
extern NSString *kCIAttributeType;
extern NSString *kCIAttributeMin;
extern NSString *kCIAttributeMax;
extern NSString *kCIAttributeSliderMin;
extern NSString *kCIAttributeSliderMax;
extern NSString *kCIAttributeDefault;
extern NSString *kCIAttributeIdentity;
extern NSString *kCIAttributeName;
extern NSString *kCIAttributeDisplayName;
```

Constants

`kCIAttributeFilterName`

The filter name, specified as an `NSString` object.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCIAttributeFilterDisplayName`

The localized version of the filter name that is displayed in the user interface.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCIAttributeFilterCategories`

An array of filter category keys that specifies all the categories in which the filter is a member.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCIAttributeClass`

The class name of the filter.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCIAttributeType`

One of the attribute types described in "Data Type Attributes" (page 181).

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCIAttributeMin`

The minimum value for a filter parameter, specified as a floating-point value.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCIAttributeMax`

The maximum value for a filter parameter, specified as a floating-point value.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCIAttributeSliderMin`
: The minimum value, specified as a floating-point value, to use for a slider that controls input values for a filter parameter.

  Available in iOS 5.0 and later.

  Declared in `CIFilter.h`.

`kCIAttributeSliderMax`
: The maximum value, specified as a floating-point value, to use for a slider that controls input values for a filter parameter.

  Available in iOS 5.0 and later.

  Declared in `CIFilter.h`.

`kCIAttributeDefault`
: The default value, specified as a floating-point value, for a filter parameter.

  Available in iOS 5.0 and later.

  Declared in `CIFilter.h`.

`kCIAttributeIdentity`
: If supplied as a value for a parameter, the parameter has no effect on the input image.

  Available in iOS 5.0 and later.

  Declared in `CIFilter.h`.

`kCIAttributeName`
: The name of the attribute.

  Available in iOS 5.0 and later.

  Declared in `CIFilter.h`.

`kCIAttributeDisplayName`
: The localized display name of the attribute.

  Available in iOS 5.0 and later.

  Declared in `CIFilter.h`.

Discussion

Attribute keys are used for the attribute dictionary of a filter. Most entries in the attribute dictionary are optional. The attribute `kCIAttributeFilterName` (page 179) is mandatory. For a parameter, the attribute `kCIAttributeClass` (page 179) is mandatory because it specifies the class name of the filter.

A parameter of type `NSNumber` does not necessarily need the attributes `kCIAttributeMin` (page 179) and `kCIAttributeMax` (page 179). These attributes are not present when the parameter has no upper or lower bounds. For example, the Gaussian blur filter has a radius parameter with a minimum of `0` but no maximum value to indicate that all nonnegative values are valid.

Declared in
`CIFilter.h`

Data Type Attributes

Numeric data types.

```
extern NSString *kCIAttributeTypeTime;
extern NSString *kCIAttributeTypeScalar;
extern NSString *kCIAttributeTypeDistance;
extern NSString *kCIAttributeTypeAngle;
extern NSString *kCIAttributeTypeBoolean;
extern NSString *kCIAttributeTypeInteger;
extern NSString *kCIAttributeTypeCount;
```

Constants

`kCIAttributeTypeTime`

A parametric time for transitions, specified as a floating-point value in the range of `0.0` to `1.0`.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCIAttributeTypeScalar`

A scalar value.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCIAttributeTypeDistance`

A distance.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCIAttributeTypeAngle`

An angle.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCIAttributeTypeBoolean`

A Boolean value.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCIAttributeTypeInteger`

An integer value.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCIAttributeTypeCount`

A positive integer value.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

Declared in
`CIFilter.h`

Vector Quantity Attributes

Vector data types.

```
extern NSString *kCIAttributeTypePosition;
extern NSString *kCIAttributeTypeOffset;
extern NSString *kCIAttributeTypePosition3;
extern NSString *kCIAttributeTypeRectangle
```

Constants

`kCIAttributeTypePosition`

A two-dimensional location in the working coordinate space. (A 2-element vector type.)

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCIAttributeTypeOffset`

An offset. (A 2-element vector type.)

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCIAttributeTypePosition3`

A three-dimensional location in the working coordinate space. (A 3-element vector type.)

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCIAttributeTypeRectangle`

A Core Image vector that specifies the *x* and *y* values of the rectangle origin, and the width (*w*) and height (*h*) of the rectangle. The vector takes the form [*x*, *y*, *w*, *h*]. (A 4-element vector type.)

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

Declared in
`CIFilter.h`

Color Attribute Keys

Color types.

```
extern NSString *kCIAttributeTypeColor;
```

Constants

`kCIAttributeTypeColor`

A Core Image color (`CIColor` object) that specifies red, green, and blue component values.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

Declared in
`CIFilter.h`

Image Attribute Keys

Image Types

```
extern NSString *kCIAttributeTypeImage;
extern NSString *kCIAttributeTypeTransform;
```

Constants

`kCIAttributeTypeImage`

A `CIImage` object.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCIAttributeTypeTransform`

An `CGAffineTransform` is associated with attribute.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

Filter Category Keys

Categories of filters.

```
extern NSString *kCICategoryDistortionEffect;
extern NSString *kCICategoryGeometryAdjustment;
extern NSString *kCICategoryCompositeOperation;
extern NSString *kCICategoryHalftoneEffect;
extern NSString *kCICategoryColorAdjustment;
extern NSString *kCICategoryColorEffect;
extern NSString *kCICategoryTransition;
extern NSString *kCICategoryTileEffect;
extern NSString *kCICategoryGenerator;
extern NSString *kCICategoryReduction;
extern NSString *kCICategoryGradient;
extern NSString *kCICategoryStylize;
extern NSString *kCICategorySharpen;
extern NSString *kCICategoryBlur;
extern NSString *kCICategoryVideo;
extern NSString *kCICategoryStillImage;
extern NSString *kCICategoryInterlaced;
extern NSString *kCICategoryNonSquarePixels;
extern NSString *kCICategoryHighDynamicRange ;
extern NSString *kCICategoryBuiltIn;
```

Constants

`kCICategoryDistortionEffect`

A filter that reshapes an image by altering its geometry to create a 3D effect. Using distortion filters, you can displace portions of an image, apply lens effects, make a bulge in an image, and perform other operation to achieve an artistic effect.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCICategoryGeometryAdjustment`

A filter that changes the geometry of an image. Some of these filters are used to warp an image to achieve an artistic effects, but these filters can also be used to correct problems in the source image. For example, you can apply an affine transform to straighten an image that is rotated with respect to the horizon.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCICategoryCompositeOperation`

A filter operates on two image sources, using the color values of one image to operate on the other. Composite filters perform computations such as computing maximum values, minimum values, and multiplying values between input images. You can use compositing filters to add effects to an image, crop an image, and achieve a variety of other effects.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCICategoryHalftoneEffect`

A filter that simulates a variety of halftone screens, to mimic the halftone process used in print media. The output of these filters has the familiar "newspaper" look of the various dot patterns. Filters are typically named after the pattern created by the virtual halftone screen, such as circular screen or hatched screen.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCICategoryColorAdjustment`

A filter that changes color values. Color adjustment filters are used to eliminate color casts, adjust hue, and correct brightness and contrast. Color adjustment filters do not perform color management; ColorSync performs color management. You can use Quartz 2D to specify the color space associated with an image. For more information, see *Color Management Overview* and *Quartz 2D Programming Guide*.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCICategoryColorEffect`

A filter that modifies the color of an image to achieve an artistic effect. Examples of color effect filters include filters that change a color image to a sepia image or a monochrome image or that produces such effects as posterizing.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCICategoryTransition`

A filter that provides a bridge between two or more images by applying a motion effect that defines how the pixels of a source image yield to that of the destination image.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCICategoryTileEffect`

A filter that typically applies an effect to an image and then create smaller versions of the image (tiles), which are then laid out to create a pattern that's infinite in extent.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCICategoryGenerator`

A filter that generates a pattern, such as a solid color, a checkerboard, or a star shine. The generated output is typically used as input to another filter.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCICategoryReduction`

A filter that reduces image data. These filters are used to solve image analysis problems.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCICategoryGradient`

A filter that generates a fill whose color varies smoothly. Exactly how color varies depends on the type of gradient—linear, radial, or Gaussian.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCICategoryStylize`

A filter that makes a photographic image look as if it was painted or sketched. These filters are typically used alone or in combination with other filters to achieve artistic effects.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCICategorySharpen`

A filter that sharpens images, increasing the contrast between the edges in an image. Examples of sharpen filters are unsharp mask and sharpen luminance.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCICategoryBlur`

A filter that softens images, decreasing the contrast between the edges in an image. Examples of blur filters are Gaussian blur and zoom blur.

Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCICategoryVideo`
: A filter that works on video images.

  Available in iOS 5.0 and later.

  Declared in `CIFilter.h`.

`kCICategoryStillImage`
: A filter that works on still images.

  Available in iOS 5.0 and later.

  Declared in `CIFilter.h`.

`kCICategoryInterlaced`
: A filter that works on interlaced images.

  Available in iOS 5.0 and later.

  Declared in `CIFilter.h`.

`kCICategoryNonSquarePixels`
: A filter that works on non-square pixels.

  Available in iOS 5.0 and later.

  Declared in `CIFilter.h`.

`kCICategoryHighDynamicRange`
: A filter that works on high dynamic range pixels.

  Available in iOS 5.0 and later.

  Declared in `CIFilter.h`.

`kCICategoryBuiltIn`
: A filter provided by Core Image. This distinguishes built-in filters from plug-in filters.

  Available in iOS 5.0 and later.

  Declared in `CIFilter.h`.

Declared in
`CIFilter.h`

Filter Parameter Keys

Keys for input parameters to filters.

```
extern NSString *kCIOutputImageKey;
extern NSString *kCIInputBackgroundImageKey;
extern NSString *kCIInputImageKey;
extern NSString *kCIInputTimeKey;
extern NSString *kCIInputTransformKey;
extern NSString *kCIInputScaleKey;
extern NSString *kCIInputAspectRatioKey;
extern NSString *kCIInputCenterKey;
extern NSString *kCIInputRadiusKey;
extern NSString *kCIInputAngleKey;
extern NSString *kCIInputWidthKey;
extern NSString *kCIInputSharpnessKey;
extern NSString *kCIInputIntensityKey;
extern NSString *kCIInputEVKey;
extern NSString *kCIInputSaturationKey;
extern NSString *kCIInputColorKey;
extern NSString *kCIInputBrightnessKey;
extern NSString *kCIInputContrastKey;
extern NSString *kCIInputMaskImageKey;
extern NSString *kCIInputTargetImageKey;
extern NSString *kCIInputExtentKey;
extern NSString *kCIInputVersionKey;
```

Constants

`kCIOutputImageKey`

A key for the `CIImage` object produced by a filter.

Available in OS X v10.5 and later. Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCIInputBackgroundImageKey`

A key for the `CIImage` object to use as a background image.

Available in OS X v10.5 and later. Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCIInputImageKey`

A key for the `CIImage` object to use as an input image. For filters that also use a background image, this key refers to the foreground image.

Available in OS X v10.5 and later. Available in iOS 5.0 and later.

Declared in `CIFilter.h`.

`kCIInputTimeKey`

A key for z scalar value (`NSNumber`) that specifies a time.

Available in OS X v10.5 and later. Not available in iOS.

Declared in `CIFilter.h`.

`kCIInputTransformKey`

A key for an `NSAffineTransform` object that specifies a transformation to apply.

Available in OS X v10.5 and later. Not available in iOS.

Declared in `CIFilter.h`.

`kCIInputScaleKey`

A key for a scalar value (`NSNumber`) that specifies the amount of the effect.

Available in OS X v10.5 and later. Not available in iOS.

Declared in `CIFilter.h`.

`kCIInputAspectRatioKey`

A key for a scalar value (`NSNumber`) that specifies a ratio.

Available in OS X v10.5 and later. Not available in iOS.

Declared in `CIFilter.h`.

`kCIInputCenterKey`

A key for a `CIVector` object that specifies the center of the area, as *x* and *y*- coordinates, to be filtered.

Available in OS X v10.5 and later. Not available in iOS.

Declared in `CIFilter.h`.

`kCIInputRadiusKey`

A key for a scalar value (`NSNumber`) that specifies that specifies the distance from the center of an effect.

Available in OS X v10.5 and later. Not available in iOS.

Declared in `CIFilter.h`.

`kCIInputAngleKey`

A key for a scalar value (`NSNumber`) that specifies an angle.

Available in OS X v10.5 and later. Not available in iOS.

Declared in `CIFilter.h`.

`kCIInputWidthKey`

A key for a scalar value (`NSNumber`) that specifies the width of the effect.

Available in OS X v10.5 and later. Not available in iOS.

Declared in `CIFilter.h`.

`kCIInputSharpnessKey`

A key for a scalar value (`NSNumber`) that specifies the amount of sharpening to apply.

Available in OS X v10.5 and later. Not available in iOS.

Declared in `CIFilter.h`.

`kCIInputIntensityKey`

A key for a scalar value (`NSNumber`) that specifies an intensity value.

Available in OS X v10.5 and later. Not available in iOS.

Declared in `CIFilter.h`.

`kCIInputEVKey`

A key for a scalar value (`NSNumber`) that specifies how many F-stops brighter or darker the image should be.

Available in OS X v10.5 and later. Not available in iOS.

Declared in `CIFilter.h`.

`kCIInputSaturationKey`

A key for a scalar value (`NSNumber`) that specifies the amount to adjust the saturation.

Available in OS X v10.5 and later. Not available in iOS.

Declared in `CIFilter.h`.

`kCIInputColorKey`

A key for a `CIColor` object that specifies a color value.

Available in OS X v10.5 and later. Not available in iOS.

Declared in `CIFilter.h`.

`kCIInputBrightnessKey`

A key for a scalar value (`NSNumber`) that specifies a brightness level.

Available in OS X v10.5 and later. Not available in iOS.

Declared in `CIFilter.h`.

`kCIInputContrastKey`

A key for a scalar value (`NSNumber`) that specifies a contrast level.

Available in OS X v10.5 and later. Not available in iOS.

Declared in `CIFilter.h`.

`kCIInputMaskImageKey`

A key for a `CIImage` object to use as a mask.

Available in OS X v10.5 and later. Not available in iOS.

Declared in `CIFilter.h`.

`kCIInputTargetImageKey`

A key for a `CIImage` object that is the target image for a transition.

Available in OS X v10.5 and later. Not available in iOS.

Declared in `CIFilter.h`.

`kCIInputExtentKey`

A key for a `CIVector` object that specifies a rectangle that defines the extent of the effect.

Available in OS X v10.5 and later. Not available in iOS.

Declared in `CIFilter.h`.

`kCIInputVersionKey`

A key for an `NSNumber` object that specifies a version number.

Not available in OS X. Available in iOS 6.0 and later.

Declared in `CIFilter.h`.

Discussion

These keys represent some of the most commonly used input parameters. A filter can use other kinds of input parameters.

Declared in

`CIFIlter.h`

CIImage Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	Library/Frameworks/CoreImage.framework
Availability	Available in iOS 5.0 and later.
Declared in	CIImage.h CIImageProvider.h
Companion guide	Core Image Programming Guide
Related sample code	AirDrop Examples Core Image Filters with Photos and Video for iOS PocketCoreImage SquareCam

Overview

The `CIImage` class represents an image. Core Image images are immutable. You use `CIImage` objects in conjunction with other Core Image classes—such as `CIFilter`, `CIContext`, `CIVector`, and `CIColor`—to take advantage of the built-in Core Image filters when processing images. You can create `CIImage` objects with data supplied from a variety of sources, including Quartz 2D images, Core Video image buffers (`CVImageBufferRef`), URL-based objects, and `NSData` objects.

Although a `CIImage` object has image data associated with it, it is not an image. You can think of a `CIImage` object as an image "recipe." A `CIImage` object has all the information necessary to produce an image, but Core Image doesn't actually render an image until it is told to do so. This "lazy evaluation" method allows Core Image to operate as efficiently as possible.

`CIContext` and `CIImage` objects are immutable, which means each can be shared safely among threads. Multiple threads can use the same GPU or CPU `CIContext` object to render `CIImage` objects. However, this is not the case for `CIFilter` objects, which are mutable. A `CIFilter` object cannot be shared safely among threads. If you app is multithreaded, each thread must create its own `CIFilter` objects. Otherwise, your app could behave unexpectedly.

Core Image also provides autoadjustment methods that analyze an image for common deficiencies and return a set of filters to correct those deficiencies. The filters are preset with values for improving image quality by altering values for skin tones, saturation, contrast, and shadows and for removing red-eye or other artifacts caused by flash. (See `“Getting Autoadjustment Filters”` (page 195).)

For a discussion of all the methods you can use to create `CIImage` objects on iOS and OS X, see *Core Image Programming Guide* .

Tasks

Creating an Image

`+ emptyImage` (page 196)
Creates and returns an empty image object.

`+ imageWithColor:` (page 198)
Creates and returns an image of infinite extent that is initialized the specified color.

`+ imageWithBitmapData:bytesPerRow:size:format:colorSpace:` (page 196)
Creates and returns an image object from bitmap data.

`+ imageWithCGImage:` (page 197)
Creates and returns an image object from a Quartz 2D image.

`+ imageWithCGImage:options:` (page 197)
Creates and returns an image object from a Quartz 2D image using the specified options.

`+ imageWithContentsOfURL:` (page 199)
Creates and returns an image object from the contents of a file.

`+ imageWithContentsOfURL:options:` (page 199)
Creates and returns an image object from the contents of a file, using the specified options.

`+ imageWithCVPixelBuffer:` (page 200)
Creates and returns an image object from the contents of `CVPixelBuffer` object.

+ `imageWithCVPixelBuffer:options:` (page 200)
  Creates and returns an image object from the contents of `CVPixelBuffer` object, using the specified options.

+ `imageWithData:` (page 201)
  Creates and returns an image object initialized with the supplied image data.

+ `imageWithData:options:` (page 202)
  Creates and returns an image object initialized with the supplied image data, using the specified options.

+ `imageWithTexture:size:flipped:colorSpace:` (page 202)
  Creates and returns an image object initialized with data supplied by an OpenGL texture.

Creating an Image by Modifying an Existing Image

– `imageByApplyingTransform:` (page 205)
  Returns a new image that represents the original image after applying an affine transform.

– `imageByCroppingToRect:` (page 206)
  Returns a new image that represents the original image after cropping to a rectangle.

Initializing an Image

– `initWithColor:` (page 208)
  Initializes an image with the specified color.

– `initWithBitmapData:bytesPerRow:size:format:colorSpace:` (page 206)
  Initializes an image object with bitmap data.

– `initWithCGImage:` (page 207)
  Initializes an image object with a Quartz 2D image.

– `initWithImage:` (page 212)
  Initializes an image object with the specified UIKit image object.

– `initWithImage:options:` (page 213)
  Initializes an image object with the specified UIKit image object, using the specified options.

– `initWithContentsOfURL:` (page 209)
  Initializes an image object by reading an image from a URL.

– `initWithContentsOfURL:options:` (page 209)
  Initializes an image object by reading an image from a URL, using the specified options.

– `initWithCVPixelBuffer:` (page 210)
  Initializes an image object from the contents of `CVPixelBuffer` object.

– `initWithCVPixelBuffer:options:` (page 210)
  Initializes an image object from the contents of `CVPixelBuffer` object using the specified options.

– `initWithData:` (page 211)
  Initializes an image object with the supplied image data.

– `initWithData:options:` (page 212)
  Initializes an image object with the supplied image data, using the specified options.

– `initWithTexture:size:flipped:colorSpace:` (page 213)
  Initializes an image object with data supplied by an OpenGL texture.

– `initWithCGImage:options:` (page 207) Deprecated in iOS 6.0
  Initializes an image object with a Quartz 2D image, using the specified options.

Getting Image Information

– `extent` (page 205)
  Returns a rectangle that specifies the extent of the image.

– `properties` (page 214)
  Returns a dictionary containing image metadata.

Getting Autoadjustment Filters

– `autoAdjustmentFilters` (page 204)
  Returns all possible automatically selected and configured filters for adjusting the image.

– `autoAdjustmentFiltersWithOptions:` (page 204)
  Returns a subset of automatically selected and configured filters for adjusting the image.

Working with Filter Regions of Interest

– `regionOfInterestForImage:inRect:` (page 215)
  Returns the region of interest for the filter chain that generates the image.

Class Methods

emptyImage

Creates and returns an empty image object.

```
+ (CIImage *)emptyImage
```

Return Value
An image object.

Availability
Available in iOS 5.0 and later.

Related Sample Code
Core Image Filters with Photos and Video for iOS

Declared in
`CIImage.h`

imageWithBitmapData:bytesPerRow:size:format:colorSpace:

Creates and returns an image object from bitmap data.

```
+ (CIImage *)imageWithBitmapData:(NSData *)d bytesPerRow:(size_t)bpr size:(CGSize)size
 format:(CIFormat)f colorSpace:(CGColorSpaceRef)cs
```

Parameters

`d`

The bitmap data for the image. This data must be premultiplied.

`bpr`

The number of bytes per row.

`size`

The dimensions of the image.

`f`

The format and size of each pixel. You must supply a pixel format constant. See “`Pixel Formats`” (page 216).

`cs`

The color space that the image is defined in. If this value is `nil`, the image is not color matched. Pass `nil` for images that don't contain color data (such as elevation maps, normal vector maps, and sampled function tables).

Return Value

An image object.

Availability

Available in iOS 5.0 and later.

See Also

`– initWithBitmapData:bytesPerRow:size:format:colorSpace:` (page 206)

Declared in

`CIImage.h`

imageWithCGImage:

Creates and returns an image object from a Quartz 2D image.

```
+ (CIImage *)imageWithCGImage:(CGImageRef)image
```

Parameters

`image`

A Quartz 2D image (`CGImageRef`) object. For more information, see *Quartz 2D Programming Guide* and *CGImage Reference*.

Return Value

An image object initialized with the contents of the Quartz 2D image.

Availability

Available in iOS 5.0 and later.

See Also

`+ imageWithCGImage:options:` (page 197)
`– initWithCGImage:` (page 207)

Related Sample Code

AirDrop Examples
Core Image Filters with Photos and Video for iOS

Declared in

`CIImage.h`

imageWithCGImage:options:

Creates and returns an image object from a Quartz 2D image using the specified options.

```
+ (CIImage *)imageWithCGImage:(CGImageRef)image options:(NSDictionary *)d
```

Parameters

`image`

A Quartz 2D image (`CGImageRef`) object. For more information, see *Quartz 2D Programming Guide* and *CGImage Reference*.

`d`

A dictionary specifying image options. (See "`Image Dictionary Keys`" (page 217).)

Return Value

An image object initialized with the contents of the Quartz 2D image and the specified options.

Availability

Available in iOS 5.0 and later.

See Also

`+ imageWithCGImage:` (page 197)
`– initWithCGImage:options:` (page 207)

Declared in

`CIImage.h`

imageWithColor:

Creates and returns an image of infinite extent that is initialized the specified color.

```
+ (CIImage *)imageWithColor:(CIColor *)color
```

Parameters

`color`

A color object.

Return Value

The image object initialized with the color represented by the `CIColor` object.

Availability

Available in iOS 5.0 and later.

See Also

`– initWithColor:` (page 208)

Related Sample Code

Core Image Filters with Photos and Video for iOS

Declared in
`CIImage.h`

imageWithContentsOfURL:

Creates and returns an image object from the contents of a file.

```
+ (CIImage *)imageWithContentsOfURL:(NSURL *)url
```

Parameters

`url`
The location of the file.

Return Value
An image object initialized with the contents of the file.

Availability
Available in iOS 5.0 and later.

See Also
`+ imageWithContentsOfURL:options:` (page 199)
`– initWithContentsOfURL:` (page 209)

Declared in
`CIImage.h`

imageWithContentsOfURL:options:

Creates and returns an image object from the contents of a file, using the specified options.

```
+ (CIImage *)imageWithContentsOfURL:(NSURL *)url options:(NSDictionary *)d
```

Parameters

`url`
The location of the file.

`d`
A dictionary specifying image options. (See “`Image Dictionary Keys`” (page 217).)

Return Value
An image object initialized with the contents of the file and set up with the specified options.

Availability
Available in iOS 5.0 and later.

See Also
`+ imageWithContentsOfURL:` (page 199)
`– initWithContentsOfURL:options:` (page 209)

Declared in
`CIImage.h`

imageWithCVPixelBuffer:

Creates and returns an image object from the contents of `CVPixelBuffer` object.

```
+ (CIImage *)imageWithCVPixelBuffer:(CVPixelBufferRef)buffer
```

Parameters
`buffer`
A `CVPixelBuffer` object.

Return Value
An image object initialized with the contents of the image buffer object.

Availability
Available in iOS 5.0 and later.

See Also
`+ imageWithCVPixelBuffer:options:` (page 200)
`– initWithCVPixelBuffer:` (page 210)

Declared in
`CIImage.h`

imageWithCVPixelBuffer:options:

Creates and returns an image object from the contents of `CVPixelBuffer` object, using the specified options.

```
+ (CIImage *)imageWithCVPixelBuffer:(CVPixelBufferRef)buffer options:(NSDictionary
*)dict
```

Parameters

`buffer`

A `CVPixelBuffer` object.

`dict`

A dictionary specifying image options. (See `“Image Dictionary Keys”` (page 217).)

Return Value

An image object initialized with the contents of the image buffer object and set up with the specified options.

Availability

Available in iOS 5.0 and later.

See Also

`+ imageWithCVPixelBuffer:` (page 200)
`– initWithCVPixelBuffer:options:` (page 210)

Related Sample Code
Core Image Filters with Photos and Video for iOS

Declared in

`CIImage.h`

imageWithData:

Creates and returns an image object initialized with the supplied image data.

```
+ (CIImage *)imageWithData:(NSData *)data
```

Parameters

`data`

The data object that holds the contents of an image file (such as TIFF, GIF, JPG, or whatever else the system supports). The image data must be premultiplied.

Return Value

An image object initialized with the supplied data, or `nil` if the method cannot create an image representation from the contents of the supplied data object.

Availability

Available in iOS 5.0 and later.

See Also

`+ imageWithData:options:` (page 202)
`– initWithData:` (page 211)

Declared in
`CIImage.h`

imageWithData:options:

Creates and returns an image object initialized with the supplied image data, using the specified options.

```
+ (CIImage *)imageWithData:(NSData *)data options:(NSDictionary *)d
```

Parameters

`data`

A pointer to the image data. The data must be premultiplied

`d`

A dictionary specifying image options. (See “`Image Dictionary Keys`” (page 217).)

Return Value
An image object initialized with the supplied data and set up with the specified options.

Availability
Available in iOS 5.0 and later.

See Also
`+ imageWithData:` (page 201)
`– initWithData:options:` (page 212)

Declared in
`CIImage.h`

imageWithTexture:size:flipped:colorSpace:

Creates and returns an image object initialized with data supplied by an OpenGL texture.

```
+ (CIImage *)imageWithTexture:(unsigned int)name size:(CGSize)size flipped:(BOOL)flag
 colorSpace:(CGColorSpaceRef)cs
```

Parameters

`name`

An OpenGL texture. Because `CIImage` objects are immutable, the texture must remain unchanged for the life of the image object. See the discussion for more information.

`size`

The dimensions of the texture.

`flag`

`YES` to have Core Image flip the coordinates of the texture vertically to convert between OpenGL and Core Image coordinate systems.

`cs`

The color space that the image is defined in. If the `colorSpace` value is `nil`, the image is not color matched. Pass `nil` for images that don't contain color data (such as elevation maps, normal vector maps, and sampled function tables).

Return Value

An image object initialized with the texture data.

Discussion

When using a texture to create a `CIImage` object, the texture must be valid in the Core Image context (`CIContext`) that you draw the `CIImage` object into. This means that one of the following must be true:

- The texture must be created using the `CGLContext` object that the `CIContext` is based on.
- The context that the texture was created in must be shared with the `CGLContext` that the `CIContext` is based on.

Note that textures do not have a retain and release mechanism. This means that your application must make sure that the texture exists for the life cycle of the image. When you no longer need the image, you can delete the texture.

Core Image ignores the texture filtering and wrap modes (`GL_TEXTURE_FILTER` and `GL_TEXTURE_WRAP`) that you set through OpenGL. The filter and wrap modes are overridden by what the CISampler object specifies when you apply a filter to the `CIImage` object.

Availability

Available in iOS 6.0 and later.

See Also

– `initWithTexture:size:flipped:colorSpace:` (page 213)

Declared in

`CIImage.h`

Instance Methods

autoAdjustmentFilters

Returns all possible automatically selected and configured filters for adjusting the image.

```
- (NSArray *)autoAdjustmentFilters
```

Return Value
An array of `CIFilter` instances preconfigured for correcting deficiencies in the supplied image.

Availability
Available in iOS 5.0 and later.

Declared in
`CIImage.h`

autoAdjustmentFiltersWithOptions:

Returns a subset of automatically selected and configured filters for adjusting the image.

```
- (NSArray *)autoAdjustmentFiltersWithOptions:(NSDictionary *)dict
```

Parameters
`options`

You can control which filters are returned by supplying one or more of the keys described in "Autoadjustment Keys" (page 217).

The options dictionary can also contain a `CIDetectorImageOrientation` key. Because some autoadjustment filters rely on face detection, you should specify an image orientation if you want to enable these filters for an image containing face whose orientation does not match that of the image.

Return Value
An array of `CIFilter` instances preconfigured for correcting deficiencies in the supplied image.

Availability
Available in iOS 5.0 and later.

Declared in
`CIImage.h`

extent

Returns a rectangle that specifies the extent of the image.

```
- (CGRect)extent
```

Return Value
A rectangle that specifies the extent of the image in working space coordinates.

Availability
Available in iOS 5.0 and later.

Related Sample Code
AirDrop Examples

Core Image Filters with Photos and Video for iOS

Declared in
`CIImage.h`

imageByApplyingTransform:

Returns a new image that represents the original image after applying an affine transform.

```
- (CIImage *)imageByApplyingTransform:(CGAffineTransform)matrix
```

Parameters
`matrix`

An affine transform.

Return Value
The transformed image object.

Availability
Available in iOS 5.0 and later.

See Also
`– imageByCroppingToRect:` (page 206)

Related Sample Code
Core Image Filters with Photos and Video for iOS

Declared in
`CIImage.h`

imageByCroppingToRect:

Returns a new image that represents the original image after cropping to a rectangle.

```
- (CIImage *)imageByCroppingToRect:(CGRect)r
```

Return Value

An image object cropped to the specified rectangle.

Availability

Available in iOS 5.0 and later.

See Also

`– imageByApplyingTransform:` (page 205)

Related Sample Code
Core Image Filters with Photos and Video for iOS

Declared in

`CIImage.h`

initWithBitmapData:bytesPerRow:size:format:colorSpace:

Initializes an image object with bitmap data.

```
- (id)initWithBitmapData:(NSData *)d bytesPerRow:(size_t)bpr size:(CGSize)size
format:(CIFormat)f colorSpace:(CGColorSpaceRef)c
```

Parameters

`d`

The bitmap data to use for the image. The data you supply must be premultiplied.

`bpr`

The number of bytes per row.

`size`

The size of the image data.

`f`

A pixel format constant. See `"Pixel Formats"` (page 216).

`c`

The color space that the image is defined in and must be a Quartz 2D color space (`CGColorSpaceRef`). Pass `nil` for images that don't contain color data (such as elevation maps, normal vector maps, and sampled function tables).

Return Value

The initialized image object or `nil` if the object could not be initialized.

Availability

Available in iOS 5.0 and later.

See Also

`+ imageWithBitmapData:bytesPerRow:size:format:colorSpace:` (page 196)

Declared in

`CIImage.h`

initWithCGImage:

Initializes an image object with a Quartz 2D image.

```
- (id)initWithCGImage:(CGImageRef)image
```

Parameters

`image`

A Quartz 2D image (`CGImageRef`) object. For more information, see *Quartz 2D Programming Guide* and *CGImage Reference*.

Return Value

The initialized image object or `nil` if the object could not be initialized.

Availability

Available in iOS 5.0 and later.

See Also

`– initWithCGImage:options:` (page 207)
`+ imageWithCGImage:` (page 197)

Declared in

`CIImage.h`

initWithCGImage:options:

Initializes an image object with a Quartz 2D image, using the specified options.

```
- (id)initWithCGImage:(CGImageRef)image options:(NSDictionary *)d
```

Parameters

`image`

A Quartz 2D image (`CGImageRef`) object. For more information, see *Quartz 2D Programming Guide* and *CGImage Reference*.

`d`

A dictionary specifying image options. (See "`Image Dictionary Keys`" (page 217).)

Return Value

The initialized image object or `nil` if the object could not be initialized.

Availability

Available in iOS 5.0 and later.

See Also

`– initWithCGImage:` (page 207)
`+ imageWithCGImage:options:` (page 197)

Related Sample Code
PocketCoreImage

Declared in

`CIImage.h`

initWithColor:

Initializes an image with the specified color.

```
- (id)initWithColor:(CIColor *)color
```

Parameters

`color`

A color object.

Return Value

The initialized image object or `nil` if the object could not be initialized.

Availability

Available in iOS 5.0 and later.

See Also

`+ imageWithColor:` (page 198)

Declared in
`CIImage.h`

initWithContentsOfURL:

Initializes an image object by reading an image from a URL.

```
- (id)initWithContentsOfURL:(NSURL *)url
```

Parameters
`url`

The location of the image file to read.

Return Value
The initialized image object or `nil` if the object could not be initialized.

Availability
Available in iOS 5.0 and later.

See Also
`– initWithContentsOfURL:options:` (page 209)
`+ imageWithContentsOfURL:` (page 199)

Declared in
`CIImage.h`

initWithContentsOfURL:options:

Initializes an image object by reading an image from a URL, using the specified options.

```
- (id)initWithContentsOfURL:(NSURL *)url options:(NSDictionary *)d
```

Parameters
`url`

The location of the image file to read.

`d`

A dictionary specifying image options. (See “`Image Dictionary Keys`” (page 217).)

Return Value
The initialized image object or `nil` if the object could not be initialized.

Availability
Available in iOS 5.0 and later.

See Also
– `initWithContentsOfURL:` (page 209)
+ `imageWithContentsOfURL:options:` (page 199)

Declared in
`CIImage.h`

initWithCVPixelBuffer:

Initializes an image object from the contents of `CVPixelBuffer` object.

```
- (id)initWithCVPixelBuffer:(CVPixelBufferRef)buffer
```

Parameters
`buffer`
A `CVPixelBuffer` object.

Return Value
The initialized image object or `nil` if the object could not be initialized.

Availability
Available in iOS 5.0 and later.

See Also
– `initWithCVPixelBuffer:options:` (page 210)
+ `imageWithCVPixelBuffer:` (page 200)

Declared in
`CIImage.h`

initWithCVPixelBuffer:options:

Initializes an image object from the contents of `CVPixelBuffer` object using the specified options.

```
- (id)initWithCVPixelBuffer:(CVPixelBufferRef)buffer options:(NSDictionary *)dict
```

Parameters
`buffer`
A `CVPixelBuffer` object.

`dict`

A dictionary that contains options for creating an image object. (See `"Image Dictionary Keys"` (page 217).) The pixel format is supplied by the `CVPixelBuffer` object.

Return Value

The initialized image object or `nil` if the object could not be initialized.

Availability

Available in iOS 5.0 and later.

See Also

`– initWithCVPixelBuffer:` (page 210)
`– initWithCVPixelBuffer:options:` (page 210)

Related Sample Code
SquareCam

Declared in

`CIImage.h`

initWithData:

Initializes an image object with the supplied image data.

```
- (id)initWithData:(NSData *)data
```

Parameters

`data`

The image data. The data you supply must be premultiplied.

Return Value

The initialized image object or `nil` if the object could not be initialized.

Availability

Available in iOS 5.0 and later.

See Also

`– initWithData:options:` (page 212)
`+ imageWithData:` (page 201)

Declared in

`CIImage.h`

initWithData:options:

Initializes an image object with the supplied image data, using the specified options.

```
- (id)initWithData:(NSData *)data options:(NSDictionary *)d
```

Parameters

`data`

The image data. The data you supply must be premultiplied.

`d`

A dictionary specifying image options. (See “`Image Dictionary Keys`” (page 217).)

Return Value

The initialized image object or `nil` if the object could not be initialized.

Availability

Available in iOS 5.0 and later.

See Also

`– initWithData:` (page 211)
`+ imageWithData:options:` (page 202)

Declared in

`CIImage.h`

initWithImage:

Initializes an image object with the specified UIKit image object.

```
- (id)initWithImage:(UIImage *)image
```

Parameters

`image`

An image containing the source data.

Return Value

The initialized image object or `nil` if the object could not be initialized.

Availability

Available in iOS 5.0 and later.

Declared in

`UIImage.h`

initWithImage:options:

Initializes an image object with the specified UIKit image object, using the specified options.

```
- (id)initWithImage:(UIImage *)image options:(NSDictionary *)options
```

Parameters

`image`

An image containing the source data.

`options`

A dictionary that contains options for creating an image object. You can supply such options as a pixel format and a color space. See "`Image Dictionary Keys`" (page 217).

Return Value

The initialized image object or `nil` if the object could not be initialized.

Availability

Available in iOS 5.0 and later.

Declared in

`UIImage.h`

initWithTexture:size:flipped:colorSpace:

Initializes an image object with data supplied by an OpenGL texture.

```
- (id)initWithTexture:(unsigned int)name size:(CGSize)size flipped:(BOOL)flag
colorSpace:(CGColorSpaceRef)cs
```

Parameters

`name`

An OpenGL texture. Because `CIImage` objects are immutable, the texture must remain unchanged for the life of the image object. See the discussion for more information.

`size`

The dimensions of the texture.

`flag`

`YES` to have Core Image flip the coordinates of the texture vertically to convert between OpenGL and Core Image coordinate systems.

`cs`
: The color space that the image is defined in. This must be a Quartz color space (`CGColorSpaceRef`). If the `colorSpace` value is `nil`, the image is not color matched. Pass `nil` for images that don't contain color data (such as elevation maps, normal vector maps, and sampled function tables).

Return Value
The initialized image object or `nil` if the object could not be initialized.

Discussion
When using a texture to create a `CIImage` object, the texture must be valid in the Core Image context (`CIContext`) that you draw the `CIImage` object into. This means that one of the following must be true:

- The texture must be created using the `CGLContext` object that the `CIContext` is based on.
- The context that the texture was created in must be shared with the `CGLContext` that the `CIContext`is based on.

Note that textures do not have a retain and release mechanism. This means that your application must make sure that the texture exists for the life cycle of the image. When you no longer need the image, you can delete the texture.

Core Image ignores the texture filtering and wrap modes (`GL_TEXTURE_FILTER` and `GL_TEXTURE_WRAP`) that you set through OpenGL. The filter and wrap modes are overridden by what the CISampler object specifies when you apply a filter to the `CIImage` object.

Availability
Available in iOS 6.0 and later.

See Also
`+ imageWithTexture:size:flipped:colorSpace:` (page 202)

Declared in
`CIImage.h`

properties

Returns a dictionary containing image metadata.

```
- (NSDictionary *)properties
```

Return Value
An `NSDictionary` object containing image metadata.

Discussion

If the `CIImage` object is the output of a filter (or filter chain), this method returns the metadata from the filter's original input image.

Availability

Available in iOS 5.0 and later.

Declared in

`CIImage.h`

regionOfInterestForImage:inRect:

Returns the region of interest for the filter chain that generates the image.

```
- (CGRect)regionOfInterestForImage:(CIImage *)im inRect:(CGRect)r
```

Parameters

`im`

Another image that is part of the filter chain that generates the image.

`r`

A rectangle in the image's coordinate space.

Return Value

A rectangle in the coordinate space of the input image (the `im` parameter).

Discussion

The region of interest is the rectangle containing pixel data in a source image (the `im` parameter) necessary to produce a corresponding rectangle in the output image. If the image is not the output of a filter (or of a chain or graph of several `CIFilter` objects), or the image in the `im` parameter is not an input to that filter, the rectangle returned is the same as that in the `r` parameter.

For example,

- If the image is the output of a filter that doubles the size of its input image, the rectangle returned will be half the size of that in the `r` parameter. (Upscaling causes every pixel in the input image to correspond to multiple pixels in the output image.)
- If the image is the output of a blur filter, the rectangle returned will be slightly larger than that in the `r` parameter. (In a blur filter, each pixel in the output image is produced using information from the corresponding pixel and those immediately surrounding it in the input image.)

Availability

Available in iOS 6.0 and later.

Declared in
`CIImage.h`

Constants

CIFormat

A data type used for specifying image pixel formats.

```
typedef int CIFormat;
```

Availability
Available in iOS 5.0 and later.

Declared in
`CIImage.h`

Pixel Formats

Image data pixel formats.

```
extern CIFormat kCIFormatARGB8;
extern CIFormat kCIFormatBGRA8;
extern CIFormat kCIFormatRGBA8;
```

Constants

`kCIFormatARGB8`

A 32 bit-per-pixel, fixed-point pixel format in which the alpha value precedes the red, green and blue color components.

Available in iOS 6.0 and later.

Declared in `CIImage.h`.

`kCIFormatBGRA8`

A 32 bit-per-pixel, fixed-point pixel format in which the blue, green, and red color components precede the alpha value.

Available in iOS 5.0 and later.

Declared in `CIImage.h`.

`kCIFormatRGBA8`

A 32 bit-per-pixel, fixed-point pixel format in which the red, green, and blue color components precede the alpha value.

Available in iOS 5.0 and later.

Declared in `CIImage.h`.

Declared in
`CIImage.h`

Image Dictionary Keys

Constants used as keys in the options dictionary when initializing an image.

```
extern NSString *kCIImageColorSpace;
extern NSString *kCIImageProperties;
```

Constants

`kCIImageColorSpace`

The key for a color space. The value you supply for this dictionary key must be a `CGColorSpaceRef` data type. For more information on this data type see *CGColorSpace Reference*. Typically you use this option when you need to load an elevation, mask, normal vector, or RAW sensor data directly from a file without color correcting it. This constant specifies to override Core Image, which, by default, assumes that data is in GenericRGB.

Available in iOS 5.0 and later.

Declared in `CIImage.h`.

`kCIImageProperties`

The key for image metadata properties. If a value for this key isn't supplied, the image's `colorSpace` dictionary are populated automatically by calling `CGImageSourceCopyPropertiesAtIndex`. To ensure that an image has no metadata properties, set the value of this key to `[NSNull null]`.

Available in iOS 5.0 and later.

Declared in `CIImage.h`.

Declared in
`CIImage.h`

Autoadjustment Keys

Constants used as keys in the options dictionary for the `autoAdjustmentFiltersWithOptions:` *(page 204) method.*

```
NSString *kCIImageAutoAdjustEnhance;
NSString *kCIImageAutoAdjustRedEye;
NSString *kCIImageAutoAdjustFeatures;
```

Constants

`kCIImageAutoAdjustEnhance`

A key used to specify whether to return enhancement filters.

The value associated with this key is a `CFBoolean` value. Supply `false` to indicate not to return enhancement filters. If you don't specify this option, Core Image assumes its value is `true`.

Available in iOS 5.0 and later.

Declared in `CIImage.h`.

`kCIImageAutoAdjustRedEye`

A key used to specify whether to return a red eye filter.

The value associated with this key is a `CFBoolean` value. Supply `false` to indicate not to return a red eye filter. If you don't specify this option, Core Image assumes its value is `true`.

Available in iOS 5.0 and later.

Declared in `CIImage.h`.

`kCIImageAutoAdjustFeatures`

A key used to specify an array of features that you want to apply enhancement and red eye filters to.

The associated value is an array of `CIFeature` objects. If you don't supply an array, the Core Image searches for features using the `CIDetector` class.

Available in iOS 5.0 and later.

Declared in `CIImage.h`.

CIVector Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	Library/Frameworks/CoreImage.framework
Availability	Available in iOS 5.0 and later.
Declared in	CIVector.h
Companion guide	Core Image Programming Guide
Related sample code	Core Image Filters with Photos and Video for iOS

Overview

The `CIVector` class is used for coordinate values and direction vectors. You typically use a `CIVector` object to pass parameter values to Core Image filters. `CIVector` objects work in conjunction with other Core Image classes, such as `CIFilter`, `CIContext`, `CIImage`, and `CIColor`, to process images using the Core Image framework.

Tasks

Creating a Vector

`+ vectorWithValues:count:` (page 224)
    Creates and returns a vector that is initialized with the specified values.

`+ vectorWithX:` (page 224)
    Creates and returns a vector that is initialized with one value.

+ `vectorWithX:Y:` (page 225)
    Creates and returns a vector that is initialized with two values.

+ `vectorWithX:Y:Z:` (page 225)
    Creates and returns a vector that is initialized with three values.

+ `vectorWithX:Y:Z:W:` (page 226)
    Creates and returns a vector that is initialized with four values.

+ `vectorWithString:` (page 223)
    Creates and returns a vector that is initialized with values provided in a string representation.

+ `vectorWithCGAffineTransform:` (page 221)
    Creates and returns a vector that is initialized with values provided by a `CGAffineTransform` structure.

+ `vectorWithCGPoint:` (page 222)
    Creates and returns a vector that is initialized with values provided by a `CGPoint` structure.

+ `vectorWithCGRect:` (page 222)
    Creates and returns a vector that is initialized with values provided by a `CGRect` structure.

Initializing a Vector

– `initWithValues:count:` (page 230)
    Initializes a vector with the provided values.

– `initWithX:` (page 230)
    Initializes the first position of a vector with the provided values.

– `initWithX:Y:` (page 231)
    Initializes the first two positions of a vector with the provided values.

– `initWithX:Y:Z:` (page 231)
    Initializes the first three positions of a vector with the provided values.

– `initWithX:Y:Z:W:` (page 232)
    Initializes four positions of a vector with the provided values.

– `initWithString:` (page 230)
    Initializes a vector with values provided in a string representation.

– `initWithCGAffineTransform:` (page 228)
    Initializes a vector that is initialized with values provided by a `CGAffineTransform` structure.

– `initWithCGPoint:` (page 229)
    Initializes a vector that is initialized with values provided by a `CGPoint` structure.

– `initWithCGRect:` (page 229)
    Initializes a vector that is initialized with values provided by a `CGRect` structure.

Getting Values From a Vector

– `valueAtIndex:` (page 233)
    Returns a value from a specific position in a vector.

– `count` (page 228)
    Returns the number of items in a vector.

– `X` (page 234)
    Returns the value located in the first position in a vector.

– `Y` (page 234)
    Returns the value located in the second position in a vector.

– `Z` (page 235)
    Returns the value located in the third position in a vector.

– `W` (page 233)
    Returns the value located in the fourth position in a vector.

– `stringRepresentation` (page 232)
    Returns a string representation for a vector.

– `CGAffineTransformValue` (page 227)
    Returns the values stored in the `CIVector` object as an affine transform.

– `CGPointValue` (page 227)
    Returns the values stored in the `CIVector` object as a point.

– `CGRectValue` (page 227)
    Returns the values stored in the `CIVector` object as an rect.

Class Methods

vectorWithCGAffineTransform:

Creates and returns a vector that is initialized with values provided by a `CGAffineTransform` structure.

```
+ (CIVector *)vectorWithCGAffineTransform:(CGAffineTransform)t
```

Parameters

`t`

A transform.

Return Value

A vector initialized with the specified values.

Discussion

The six values that comprise the affine transform fill the first six positions of the resulting `CIVector` object.

Availability

Available in iOS 5.0 and later.

Declared in

`CIVector.h`

vectorWithCGPoint:

Creates and returns a vector that is initialized with values provided by a `CGPoint` structure.

```
+ (CIVector *)vectorWithCGPoint:(CGPoint)p
```

Parameters

`p`

A point.

Return Value

A vector initialized with the specified values.

Discussion

The `CGPoint` structure's X and Y values are stored in the vector's X and Y properties.

Availability

Available in iOS 5.0 and later.

Declared in

`CIVector.h`

vectorWithCGRect:

Creates and returns a vector that is initialized with values provided by a `CGRect` structure.

```
+ (CIVector *)vectorWithCGRect:(CGRect)r
```

Parameters

`r`

A rect.

Return Value

A vector initialized with the specified values.

Discussion

The `CGRect` structure's X, Y, height and width values are stored in the vector's X, Y, Z and W properties.

Availability

Available in iOS 5.0 and later.

Declared in

`CIVector.h`

vectorWithString:

Creates and returns a vector that is initialized with values provided in a string representation.

```
+ (CIVector *)vectorWithString:(NSString *)representation
```

Parameters

`representation`

A string that is in one of the formats returned by the `stringRepresentation` method.

Discussion

Some typical string representations for vectors are:

```
@"[1.0 0.5 0.3]"
```

which specifies a `vec3` vector whose components are `X = 1.0`, `Y = 0.5`, and `Z = 0.3`

```
@"[10.0 23.0]
```

which specifies a `vec2` vector show components are `X = 10.0` and `Y = 23.0`

Availability

Available in iOS 5.0 and later.

See Also

– `stringRepresentation` (page 232)

Declared in
`CIVector.h`

vectorWithValues:count:

Creates and returns a vector that is initialized with the specified values.

```
+ (CIVector *)vectorWithValues:(const CGFloat *)values count:(size_t)count
```

Parameters

`values`
    The values to initialize the vector with.

`count`
    The number of values in the vector.

Return Value
A vector initialized with the provided values.

Availability
Available in iOS 5.0 and later.

Related Sample Code
Core Image Filters with Photos and Video for iOS

Declared in
`CIVector.h`

vectorWithX:

Creates and returns a vector that is initialized with one value.

```
+ (CIVector *)vectorWithX:(CGFloat)x
```

Parameters

`x`
    The value to initialize the vector with.

Return Value
A vector initialized with the specified value.

Availability
Available in iOS 5.0 and later.

Declared in
`CIVector.h`

vectorWithX:Y:

Creates and returns a vector that is initialized with two values.

```
+ (CIVector *)vectorWithX:(CGFloat)x Y:(CGFloat)y
```

Parameters

`x`
: The value for the first position in the vector.

`y`
: The value for the second position in the vector.

Return Value
A vector initialized with the specified values.

Availability
Available in iOS 5.0 and later.

Related Sample Code
Core Image Filters with Photos and Video for iOS

Declared in
`CIVector.h`

vectorWithX:Y:Z:

Creates and returns a vector that is initialized with three values.

```
+ (CIVector *)vectorWithX:(CGFloat)x Y:(CGFloat)y Z:(CGFloat)z
```

Parameters

`x`
: The value for the first position in the vector.

`y`
: The value for the second position in the vector.

`z`
: The value for the third position in the vector.

Return Value

A vector initialized with the specified values.

Availability

Available in iOS 5.0 and later.

Declared in

`CIVector.h`

vectorWithX:Y:Z:W:

Creates and returns a vector that is initialized with four values.

```
+ (CIVector *)vectorWithX:(CGFloat)x Y:(CGFloat)y Z:(CGFloat)z W:(CGFloat)w
```

Parameters

`x`

  The value for the first position in the vector.

`y`

  The value for the second position in the vector.

`z`

  The value for the third position in the vector.

`w`

  The value for the fourth position in the vector.

Return Value

A vector initialized with the specified values.

Availability

Available in iOS 5.0 and later.

Related Sample Code

Core Image Filters with Photos and Video for iOS

Declared in

`CIVector.h`

Instance Methods

CGAffineTransformValue

Returns the values stored in the `CIVector` object as an affine transform.

```
- (CGAffineTransform)CGAffineTransformValue
```

Return Value
A transform.

Discussion
The first six values in the vector become the values that comprise the affine transform.

Availability
Available in iOS 5.0 and later.

Declared in
`CIVector.h`

CGPointValue

Returns the values stored in the `CIVector` object as a point.

```
- (CGPoint)CGPointValue
```

Return Value
A point.

Discussion
The vector's X and Y property values become the `CGPoint` structure's X and Y values.

Availability
Available in iOS 5.0 and later.

Declared in
`CIVector.h`

CGRectValue

Returns the values stored in the `CIVector` object as an rect.

```
- (CGRect)CGRectValue
```

Return Value

A rect.

Discussion

The vector's X, Y, Z and W property values become the `CGRect` structure's X, Y, height and width values.

Availability

Available in iOS 5.0 and later.

Declared in

`CIVector.h`

count

Returns the number of items in a vector.

```
- (size_t)count
```

Return Value

The number of items in the vector.

Availability

Available in iOS 5.0 and later.

Related Sample Code
Core Image Filters with Photos and Video for iOS

Declared in

`CIVector.h`

initWithCGAffineTransform:

Initializes a vector that is initialized with values provided by a `CGAffineTransform` structure.

```
- (id)initWithCGAffineTransform:(CGAffineTransform)r
```

Parameters

`r`

  A transform.

Discussion

The six values that comprise the affine transform fill the first six positions of the resulting `CIVector` object.

Availability
Available in iOS 5.0 and later.

Declared in
`CIVector.h`

initWithCGPoint:

Initializes a vector that is initialized with values provided by a `CGPoint` structure.

```
- (id)initWithCGPoint:(CGPoint)p
```

Parameters

`p`
    A point.

Discussion
The `CGPoint` structure's X and Y values are stored in the vector's X and Y properties.

Availability
Available in iOS 5.0 and later.

Declared in
`CIVector.h`

initWithCGRect:

Initializes a vector that is initialized with values provided by a `CGRect` structure.

```
- (id)initWithCGRect:(CGRect)r
```

Parameters

`r`
    A rect.

Discussion
The `CGRect` structure's X, Y, height and width values are stored in the vector's X, Y, Z and W properties.

Availability
Available in iOS 5.0 and later.

Declared in
`CIVector.h`

initWithString:

Initializes a vector with values provided in a string representation.

```
- (id)initWithString:(NSString *)representation
```

Parameters

`representation`

A string that is in one of the formats returned by the `stringRepresentation` method.

Availability

Available in iOS 5.0 and later.

See Also

– `stringRepresentation` (page 232)

Declared in

`CIVector.h`

initWithValues:count:

Initializes a vector with the provided values.

```
- (id)initWithValues:(const CGFloat *)values count:(size_t)count
```

Parameters

`values`

The values to initialize the vector with.

`count`

The number of values specified by the `values` argument.

Availability

Available in iOS 5.0 and later.

Declared in

`CIVector.h`

initWithX:

Initializes the first position of a vector with the provided values.

```
- (id)initWithX:(CGFloat)x
```

Parameters

`x`

The initialization value.

Availability

Available in iOS 5.0 and later.

Declared in

`CIVector.h`

initWithX:Y:

Initializes the first two positions of a vector with the provided values.

```
- (id)initWithX:(CGFloat)x Y:(CGFloat)y
```

Parameters

`x`

The initialization value for the first position.

`y`

The initialization value for the second position.

Availability

Available in iOS 5.0 and later.

Declared in

`CIVector.h`

initWithX:Y:Z:

Initializes the first three positions of a vector with the provided values.

```
- (id)initWithX:(CGFloat)x Y:(CGFloat)y Z:(CGFloat)z
```

Parameters

`x`

The initialization value for the first position.

`y`

The initialization value for the second position.

`z`

The initialization value for the third position.

Availability
Available in iOS 5.0 and later.

Declared in
`CIVector.h`

initWithX:Y:Z:W:

Initializes four positions of a vector with the provided values.

```
- (id)initWithX:(CGFloat)x Y:(CGFloat)y Z:(CGFloat)z W:(CGFloat)w
```

Parameters

`x`

The initialization value for the first position.

`y`

The initialization value for the second position.

`z`

The initialization value for the third position.

`w`

The initialization value for the fourth position.

Availability
Available in iOS 5.0 and later.

Declared in
`CIVector.h`

stringRepresentation

Returns a string representation for a vector.

```
- (NSString *)stringRepresentation
```

Return Value
A string object.

Discussion
You convert the string representation returned by this method to a vector by supplying it as a parameter to the `vectorWithString:` method.

Some typical string representations for vectors are:

`@"[1.0 0.5 0.3]"`

which specifies a `vec3` vector whose components are `X = 1.0`, `Y = 0.5`, and `Z = 0.3`

`@"[10.0 23.0]`

which specifies a `vec2` vector show components are `X = 10.0` and `Y = 23.0`

Availability
Available in iOS 5.0 and later.

See Also
`+ vectorWithString:` (page 223)

Declared in
`CIVector.h`

valueAtIndex:

Returns a value from a specific position in a vector.

```
- (CGFloat)valueAtIndex:(size_t)index
```

Parameters

`index`
    The position in the vector of the value that you want to retrieve.

Return Value
The value retrieved from the vector or `0` if the position is undefined.

Discussion
The numbering of elements in a vector begins with zero.

Availability
Available in iOS 5.0 and later.

Declared in
`CIVector.h`

W

Returns the value located in the fourth position in a vector.

```
- (CGFloat)W
```

Return Value
The value retrieved from the vector.

Availability
Available in iOS 5.0 and later.

Declared in
`CIVector.h`

X

Returns the value located in the first position in a vector.

```
- (CGFloat)X
```

Return Value
The value retrieved from the vector.

Availability
Available in iOS 5.0 and later.

Declared in
`CIVector.h`

Y

Returns the value located in the second position in a vector.

```
- (CGFloat)Y
```

Return Value
The value retrieved from the vector.

Availability
Available in iOS 5.0 and later.

Declared in
`CIVector.h`

Z

Returns the value located in the third position in a vector.

```
- (CGFloat)Z
```

Return Value
The value retrieved from the vector.

Availability
Available in iOS 5.0 and later.

Declared in
`CIVector.h`

NSValue Core Animation Additions

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in iOS 2.0 and later.
Declared in	QuartzCore/CATransform3D.h
Companion guides	Core Animation Programming Guide Core Animation Cookbook

Overview

Core Animation adds two methods to the Foundation framework's `NSValue` class to support `CATransform3D` structure values.

Tasks

Creating an NSValue

`+ valueWithCATransform3D:` (page 237)
Creates and returns an NSValue object that contains a given `CATransform3D` structure.

Accessing Data

`– CATransform3DValue` (page 237)
Returns an `CATransform3D` structure representation of the receiver.

Class Methods

valueWithCATransform3D:

Creates and returns an NSValue object that contains a given `CATransform3D` structure.

```
+ (NSValue *)valueWithCATransform3D:(CATransform3D)aTransform
```

Parameters

`aTransform`

The value for the new object.

Return Value

A new `NSValue` object that contains the value of `aTransform`.

Availability

Available in iOS 2.0 and later.

Related Sample Code
MoveMe

Declared in
`CATransform3D.h`

Instance Methods

CATransform3DValue

Returns an `CATransform3D` structure representation of the receiver.

```
- (CATransform3D)CATransform3DValue
```

Return Value

An `CATransform3D` structure representation of the receiver.

Availability

Available in iOS 2.0 and later.

Declared in
`CATransform3D.h`

Protocols

CAAction Protocol Reference

Adopted by	CAAnimation
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in iOS 2.0 and later.
Declared in	CALayer.h
Companion guides	Core Animation Programming Guide Core Animation Cookbook

Overview

The `CAAction` protocol provides an interface that allows an object to respond to an action triggered by an `CALayer`. When queried with an action identifier (a key path, an external action name, or a predefined action identifier) the layer returns the appropriate action object–which must implement the `CAAction` protocol–and sends it a `runActionForKey:object:arguments:` (page 239) message.

Tasks

Responding to an Action

– `runActionForKey:object:arguments:` (page 239)
    Called to trigger the action specified by the identifier. (required)

Instance Methods

runActionForKey:object:arguments:

Called to trigger the action specified by the identifier. (required)

```
- (void)runActionForKey:(NSString *)key
object:(id)anObject
arguments:(NSDictionary *)dict
```

Parameters

`key`

The identifier of the action. The identifier may be a key or key path relative to `anObject`, an arbitrary external action, or one of the action identifiers defined in *CALayer Class Reference* .

`anObject`

The layer on which the action should occur.

`dict`

A dictionary containing parameters associated with this event. May be `nil`.

Availability

Available in iOS 2.0 and later.

Declared in

`CALayer.h`

CAMediaTiming Protocol Reference

Adopted by	CAAnimation CALayer
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in iOS 2.0 and later.
Declared in	CAMediaTiming.h
Companion guides	Core Animation Programming Guide Core Animation Cookbook

Overview

The `CAMediaTiming` protocol models a hierarchical timing system, with each object describing the mapping of time values from the object's parent to local time.

Absolute time is defined as mach time converted to seconds. The `CACurrentMediaTime` (page 249) function is provided as a convenience for getting the current absolute time.

The conversion from parent time to local time has two stages:

1. Conversion to "active local time". This includes the point at which the object appears in the parent object's timeline and how fast it plays relative to the parent.
2. Conversion from "active local time" to "basic local time". The timing model allows for objects to repeat their basic duration multiple times and, optionally, to play backwards before repeating.

Tasks

Animation Start Time

`beginTime` (page 243) *property*
: Specifies the begin time of the receiver in relation to its parent object, if applicable. (required)

`timeOffset` (page 245) *property*
: Specifies an additional time offset in active local time. (required)

Repeating Animations

`repeatCount` (page 244) *property*
: Determines the number of times the animation will repeat. (required)

`repeatDuration` (page 244) *property*
: Determines how many seconds the animation will repeat for. (required)

Duration and Speed

`duration` (page 243) *property*
: Specifies the basic duration of the animation, in seconds. (required)

`speed` (page 245) *property*
: Specifies how time is mapped to receiver's time space from the parent time space. (required)

Playback Modes

`autoreverses` (page 243) *property*
: Determines if the receiver plays in the reverse upon completion. (required)

`fillMode` (page 244) *property*
: Determines if the receiver's presentation is frozen or removed once its active duration has completed. (required)

Properties

autoreverses

Determines if the receiver plays in the reverse upon completion. (required)

```
@property BOOL autoreverses
```

Discussion
When `YES`, the receiver plays backwards after playing forwards. Defaults to `NO`.

Availability
Available in iOS 2.0 and later.

Declared in
`CAMediaTiming.h`

beginTime

Specifies the begin time of the receiver in relation to its parent object, if applicable. (required)

```
@property CFTimeInterval beginTime
```

Discussion
Defaults to 0.

Availability
Available in iOS 2.0 and later.

Declared in
`CAMediaTiming.h`

duration

Specifies the basic duration of the animation, in seconds. (required)

```
@property CFTimeInterval duration
```

Discussion
Defaults to 0.

Availability
Available in iOS 2.0 and later.

Related Sample Code
MoveMe

Declared in
`CAMediaTiming.h`

fillMode

Determines if the receiver's presentation is frozen or removed once its active duration has completed. (required)

```
@property(copy) NSString *fillMode
```

Discussion
The possible values are described in `"Fill Modes"` (page 246). The default is `kCAFillModeRemoved` (page 246).

Availability
Available in iOS 2.0 and later.

Declared in
`CAMediaTiming.h`

repeatCount

Determines the number of times the animation will repeat. (required)

```
@property float repeatCount
```

Discussion
May be fractional. If the `repeatCount` is 0, it is ignored. Defaults to 0. If both `repeatDuration` (page 244) and `repeatCount` (page 244) are specified the behavior is undefined.

Setting this property to `HUGE_VALF` will cause the animation to repeat forever.

Availability
Available in iOS 2.0 and later.

Declared in
`CAMediaTiming.h`

repeatDuration

Determines how many seconds the animation will repeat for. (required)

```
@property CFTimeInterval repeatDuration
```

Discussion

Defaults to 0. If the `repeatDuration` is 0, it is ignored. If both `repeatDuration` (page 244) and `repeatCount` (page 244) are specified the behavior is undefined.

Availability

Available in iOS 2.0 and later.

Declared in

`CAMediaTiming.h`

speed

Specifies how time is mapped to receiver's time space from the parent time space. (required)

```
@property float speed
```

Discussion

For example, if `speed` is 2.0 local time progresses twice as fast as parent time. Defaults to 1.0.

Availability

Available in iOS 2.0 and later.

Declared in

`CAMediaTiming.h`

timeOffset

Specifies an additional time offset in active local time. (required)

```
@property CFTimeInterval timeOffset
```

Discussion

Defaults to 0. .

Availability

Available in iOS 2.0 and later.

Declared in

`CAMediaTiming.h`

Constants

Fill Modes

These constants determine how the timed object behaves once its active duration has completed. They are used with the `fillMode` (page 244) property.

```
NSString * const kCAFillModeRemoved;
NSString * const kCAFillModeForwards;
NSString * const kCAFillModeBackwards;
NSString * const kCAFillModeBoth;
NSString * const kCAFillModeFrozen;
```

Constants

`kCAFillModeRemoved`

The receiver is removed from the presentation when the animation is completed.

Available in iOS 2.0 and later.

Declared in `CAMediaTiming.h`.

`kCAFillModeForwards`

The receiver remains visible in its final state when the animation is completed.

Available in iOS 2.0 and later.

Declared in `CAMediaTiming.h`.

`kCAFillModeBackwards`

The receiver clamps values before zero to zero when the animation is completed.

Available in iOS 2.0 and later.

Declared in `CAMediaTiming.h`.

`kCAFillModeBoth`

The receiver clamps values at both ends of the object's time space

Available in iOS 2.0 and later.

Declared in `CAMediaTiming.h`.

`kCAFillModeFrozen`

The mode was deprecated before OS X v10.5 shipped.

Available in iOS 2.0 and later.

Deprecated in iOS 4.0.

Declared in `CAMediaTiming.h`.

Declared in

`CAMediaTiming.h`

Other References

Core Animation Function Reference

Framework	QuartzCore/QuartzCore.h
Declared in	CABase.h CATransform3D.h

Overview

Functions by Task

Timing Functions

`CACurrentMediaTime` (page 249)
    Returns the current absolute time, in seconds.

Transform Functions

`CATransform3DIsIdentity` (page 251)
    Returns a Boolean value that indicates whether the transform is the identity transform.

`CATransform3DEqualToTransform` (page 250)
    Returns a Boolean value that indicates whether the two transforms are exactly equal.

`CATransform3DMakeTranslation` (page 253)
    Returns a transform that translates by '(tx, ty, tz)'. t' = [1 0 0 0; 0 1 0 0; 0 0 1 0; tx ty tz 1].

`CATransform3DMakeScale` (page 252)
    Returns a transform that scales by `(sx, sy, sz)': * t' = [sx 0 0 0; 0 sy 0 0; 0 0 sz 0; 0 0 0 1].

`CATransform3DMakeRotation` (page 252)
    Returns a transform that rotates by 'angle' radians about the vector '(x, y, z)'. If the vector has length zero the identity transform is returned.

`CATransform3DTranslate` (page 254)
: Translate 't' by '(tx, ty, tz)' and return the result: t' = translate(tx, ty, tz) * t.

`CATransform3DScale` (page 253)
: Scale 't' by '(sx, sy, sz)' and return the result: t' = scale(sx, sy, sz) * t.

`CATransform3DRotate` (page 253)
: Rotate 't' by 'angle' radians about the vector '(x, y, z)' and return the result. If the vector has zero length the behavior is undefined: t' = rotation(angle, x, y, z) * t.

`CATransform3DConcat` (page 250)
: Concatenate 'b' to 'a' and return the result: t' = a * b.

`CATransform3DInvert` (page 251)
: Invert 't' and return the result. Returns the original matrix if 't' has no inverse.

`CATransform3DMakeAffineTransform` (page 252)
: Return a transform with the same effect as affine transform 'm'.

`CATransform3DIsAffine` (page 251)
: Returns true if 't' can be exactly represented by an affine transform.

`CATransform3DGetAffineTransform` (page 250)
: Returns the affine transform represented by 't'. If 't' can not be exactly represented as an affine transform the returned value is undefined.

Functions

CACurrentMediaTime

Returns the current absolute time, in seconds.

```
CFTimeInterval CACurrentMediaTime (void);
```

Return Value

A `CFTimeInterval` derived by calling `mach_absolute_time()` and converting the result to seconds.

Availability

Available in iOS 2.0 and later.

Related Sample Code
GLAirplay

Declared in

`CABase.h`

CATransform3DConcat

*Concatenate 'b' to 'a' and return the result: t' = a * b.*

```
CATransform3D CATransform3DConcat (CATransform3D a, CATransform3D b);
```

Availability

Available in iOS 2.0 and later.

Declared in

`CATransform3D.h`

CATransform3DEqualToTransform

Returns a Boolean value that indicates whether the two transforms are exactly equal.

```
bool CATransform3DEqualToTransform (CATransform3D a, CATransform3D b);
```

Return Value

`YES` if `a` and `b` are exactly equal, otherwise `NO`.

Availability

Available in iOS 2.0 and later.

Declared in

`CATransform3D.h`

CATransform3DGetAffineTransform

Returns the affine transform represented by 't'. If 't' can not be exactly represented as an affine transform the returned value is undefined.

```
CGAffineTransform CATransform3DGetAffineTransform (CATransform3D t);
```

Availability

Available in iOS 2.0 and later.

Declared in
`CATransform3D.h`

CATransform3DInvert

Invert 't' and return the result. Returns the original matrix if 't' has no inverse.

```
CATransform3D CATransform3DInvert (CATransform3D t);
```

Availability
Available in iOS 2.0 and later.

Declared in
`CATransform3D.h`

CATransform3DIsAffine

Returns true if 't' can be exactly represented by an affine transform.

```
bool CATransform3DIsAffine (CATransform3D t);
```

Availability
Available in iOS 2.0 and later.

Declared in
`CATransform3D.h`

CATransform3DIsIdentity

Returns a Boolean value that indicates whether the transform is the identity transform.

```
bool CATransform3DIsIdentity (CATransform3D t);
```

Return Value
`YES` if `t` is the identity transform, otherwise `NO`.

Availability
Available in iOS 2.0 and later.

Declared in
`CATransform3D.h`

CATransform3DMakeAffineTransform

Return a transform with the same effect as affine transform 'm'.

```
CATransform3D CATransform3DMakeAffineTransform (CGAffineTransform m)
```

Availability
Available in iOS 2.0 and later.

Declared in
`CATransform3D.h`

CATransform3DMakeRotation

Returns a transform that rotates by 'angle' radians about the vector '(x, y, z)'. If the vector has length zero the identity transform is returned.

```
CATransform3D CATransform3DMakeRotation (CGFloat angle, CGFloat x, CGFloat y, CGFloat z);
```

Availability
Available in iOS 2.0 and later.

Related Sample Code
oalTouch

Declared in
`CATransform3D.h`

CATransform3DMakeScale

*Returns a transform that scales by `(sx, sy, sz)': * t' = [sx 0 0 0; 0 sy 0 0; 0 0 sz 0; 0 0 0 1].*

```
CATransform3D CATransform3DMakeScale (CGFloat sx, CGFloat sy,
   CGFloat sz);
```

Availability
Available in iOS 2.0 and later.

Declared in
`CATransform3D.h`

CATransform3DMakeTranslation

Returns a transform that translates by '(tx, ty, tz)'. t' = [1 0 0 0; 0 1 0 0; 0 0 1 0; tx ty tz 1].

```
CATransform3D CATransform3DMakeTranslation (CGFloat tx, CGFloat ty, CGFloat tz)
```

Availability
Available in iOS 2.0 and later.

Related Sample Code
oalTouch

Declared in
`CATransform3D.h`

CATransform3DRotate

*Rotate 't' by 'angle' radians about the vector '(x, y, z)' and return the result. If the vector has zero length the behavior is undefined: t' = rotation(angle, x, y, z) * t.*

```
CATransform3D CATransform3DRotate (CATransform3D t, CGFloat angle, CGFloat x, CGFloat y,
CGFloat z)
```

Availability
Available in iOS 2.0 and later.

Declared in
`CATransform3D.h`

CATransform3DScale

*Scale 't' by '(sx, sy, sz)' and return the result: t' = scale(sx, sy, sz) * t.*

```
CATransform3D CATransform3DScale (CATransform3D t, CGFloat sx, CGFloat sy, CGFloat sz)
```

Availability
Available in iOS 2.0 and later.

Declared in
`CATransform3D.h`

CATransform3DTranslate

*Translate 't' by '(tx, ty, tz)' and return the result: t' = translate(tx, ty, tz) * t.*

```
CATransform3D CATransform3DTranslate (CATransform3D t, CGFloat tx, CGFloat ty, CGFloat tz);
```

Availability
Available in iOS 2.0 and later.

Declared in
`CATransform3D.h`

Document Revision History

This table describes the changes to *Quartz Core Framework Reference* .

Date	Notes
2012-01-09	Added CADisplayLink.
2008-03-12	Added links to missing classes.
2007-02-17	Added two Core Image documents and the Core Animation classes.
2006-05-23	First publication of this content as a collection of separate documents.