Quartz Core Framework Reference

Graphics & Animation

2009-09-09

Apple Inc.
© 2009 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, iPhone, Mac, Mac OS, Objective-C, Quartz, QuickDraw, QuickTime, and Spaces are trademarks of Apple Inc., registered in the United States and other countries.

Aperture is a trademark of Apple Inc.

Helvetica and Times are registered trademarks of Heidelberger Druckmaschinen AG, available from Linotype Library GmbH.

IOS is a trademark or registered trademark of Cisco in the U.S. and other countries and is used under license.

OpenGL is a registered trademark of Silicon Graphics, Inc.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

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 CVBase.h CVBuffer.h CVHostTime.h CVImageBuffer.h CVPixelBuffer.h CVPixelBufferPool.h CVPixelFormatDescription.h CVReturn.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

Overview

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

Tasks

Archiving Properties

`– shouldArchiveValueForKey:` (page 15)
    Specifies whether the value of the property for a given key is archived.

Providing Default Values for Properties

`+ defaultValueForKey:` (page 14)
    Specifies the default value of the property with the specified key.

Creating an Animation

+ `animation` (page 13)
  Creates and returns a new `CAAnimation` instance.

Animation Attributes

`removedOnCompletion` (page 13) *property*
  Determines if the animation is removed from the target layer's animations upon completion.

- `isRemovedOnCompletion` (page 14)
  A synthesized accessor for the `removedOnCompletion` (page 13) property.

`timingFunction` (page 13) *property*
  An optional timing function defining the pacing of the animation.

Getting and Setting the Delegate

`delegate` (page 12) *property*
  Specifies the receiver's delegate object.

Animation Progress

- `animationDidStart:` (page 15) *delegate method*
  Called when the animation begins its active duration.

- `animationDidStop:finished:` (page 15) *delegate method*
  Called when the animation completes its active duration or is removed from the object it is attached to.

Properties

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

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 *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.

Declared In
`CAAnimation.h`

removedOnCompletion

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

```
@property 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.

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.

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.

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

isRemovedOnCompletion

A synthesized accessor for the `removedOnCompletion` (page 13) property.

```
- (BOOL)isRemovedOnCompletion
```

See Also
`@property removedOnCompletion` (page 13)

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

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 18) array are currently ignored. The `CAAnimationGroup` delegate does receive these messages.

Note: The `delegate` and `removedOnCompletion` properties of animations in the `animations` (page 18) property are currently ignored.

Tasks

Grouped Animations

`animations` (page 18) *property*
An array of `CAAnimation` objects to be evaluated in the time space of the receiver.

Properties

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

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.

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

Overview

`CABasicAnimation` provides basic, single-keyframe animation capabilities for a layer property. You create an instance of `CABasicAnimation` using the inherited `animationWithKeyPath:` (page 97) method, specifying the key path of the property to be animated in the render tree.

Setting Interpolation Values

The `fromValue` (page 20), `byValue` (page 20) and `toValue` (page 21) 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 20) and `toValue` (page 21) are non-`nil`. Interpolates between `fromValue` (page 20) and `toValue` (page 21).
- `fromValue` (page 20) and `byValue` (page 20) are non-`nil`. Interpolates between `fromValue` (page 20) and (`fromValue` (page 20) + `byValue` (page 20)).
- `byValue` (page 20) and `toValue` (page 21) are non-`nil`. Interpolates between (`toValue` (page 21) - `byValue` (page 20)) and `toValue` (page 21).
- `fromValue` (page 20) is non-`nil`. Interpolates between `fromValue` (page 20) and the current presentation value of the property.

- `toValue` (page 21) is non-`nil`. Interpolates between the current value of `keyPath` in the target layer's presentation layer and `toValue` (page 21).
- `byValue` (page 20) is non-`nil`. Interpolates between the current value of `keyPath` in the target layer's presentation layer and that value plus `byValue` (page 20).
- 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 20) *property*
Defines the value the receiver uses to start interpolation.

`toValue` (page 21) *property*
Defines the value the receiver uses to end interpolation.

`byValue` (page 20) *property*
Defines the value the receiver uses to perform relative interpolation.

Properties

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

byValue

Defines the value the receiver uses to perform relative interpolation.

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

Discussion
See "Setting Interpolation Values" (page 19) 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 19) 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 19) for details on how `toValue` interacts with the other interpolation values.

Availability
Available in iOS 2.0 and later.

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

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 25) 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 24) 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 25) 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 25) property.

When your application finishes with a display link, it should call `invalidate` (page 27) 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 26)
Returns a new display link.

Scheduling the Display Link to Send Notifications

`– addToRunLoop:forMode:` (page 26)
Registers the display link with a run loop.

`– removeFromRunLoop:forMode:` (page 27)
Removes the display link from the run loop for the given mode.

`– invalidate` (page 27)
Removes the display link from all run loop modes.

Configuring the Display Link

`duration` (page 24) *property*
The time interval between screen refresh updates. (read-only)

`frameInterval` (page 25) *property*
The number of frames that must pass before the display link notifies the target again.

`paused` (page 25) *property*
A Boolean value that states whether the display link's notifications to the target are suspended.

`timestamp` (page 25) *property*
The time value associated with the last frame that was displayed. (read-only)

Properties

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

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 25).

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.

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.

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 27) message to the display link.

Availability
Available in iOS 3.1 and later.

See Also
`– removeFromRunLoop:forMode:` (page 27)

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 26)

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	aurioTouch GLSprite 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 30) *property*
The properties of the native windowing surface.

Properties

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

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

Overview

`CAKeyframeAnimation` provides generic keyframe animation capabilities for a layer property in the render tree. You create an `CAKeyframeAnimation` instance using the inherited `animationWithKeyPath:` (page 97) method, specifying the key path of the property updated in the render tree during the animation. The animation provides a series of keyframe values, either as an array or a series of points in a `CGPathRef`. While animating, it updates the value of the property in the render tree with values calculated using the specified interpolation calculation mode.

Tasks

Providing Keyframe Values

`path` (page 34) *property*
An optional `CGPathRef` that provides the keyframe values for the receiver.

`values` (page 35) *property*
An array of objects that provide the keyframe values for the receiver.

Keyframe Timing

`keyTimes` (page 33) *property*
An optional array of `NSNumber` objects that define the duration of each keyframe segment.

`timingFunctions` (page 35) *property*
An optional array of `CAMediaTimingFunction` instances that defines the pacing of the each keyframe segment.

`calculationMode` (page 33) *property*
Specifies how intermediate keyframe values are calculated by the receiver.

Rotation Mode Attribute

`rotationMode` (page 34) *property*
Determines whether objects animating along the path rotate to match the path tangent.

Cubic Mode Attributes

`tensionValues` (page 35) *property*
An array of `NSNumber` objects that define the tightness of the curve.

`continuityValues` (page 33) *property*
An array of `NSNumber` objects that define the sharpness of the timing curve's corners.

`biasValues` (page 32) *property*
An array of `NSNumber` objects that define the position of the curve relative to a control point.

Properties

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

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 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 duration of each keyframe segment.

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

Discussion

Each value in the array is a floating point number between 0.0 and 1.0 and corresponds to one element in the values array. Each element in the `keyTimes` array defines the duration of the corresponding keyframe value as a fraction of the total duration of the animation. Each element value must be greater than, or equal to, the previous value.

The appropriate values in the `keyTimes` array are dependent on the `calculationMode` (page 33) property.

- If the calculationMode is set to `kCAAnimationLinear`, the first value in the array must be 0.0 and the last value must be 1.0. Values are interpolated between the specified key times.
- If the calculationMode is set to `kCAAnimationDiscrete`, the first value in the array must be 0.0.
- If the calculationMode is set to `kCAAnimationPaced` or `kCAAnimationCubicPaced`, the `keyTimes` array is ignored.

If the values in the `keyTimes` array are invalid or inappropriate for the `calculationMode`, the `keyTimes` array is ignored.

Availability
Available in iOS 2.0 and later.

Declared In
`CAAnimation.h`

path

An optional `CGPathRef` that provides the keyframe values for the receiver.

```
@property CGPathRef path;
```

Discussion
Defaults to `nil`. Specifying a path overrides the `values` (page 35) property. Each point in the path, except for move-to points, defines a single keyframe segment for the purpose of timing and interpolation. For constant velocity animation along the path, `calculationMode` (page 33) should be set to `kCAAnimationPaced` (page 36).

Availability
Available in iOS 2.0 and later.

See Also
`@property rotationMode` (page 34)

Declared In
`CAAnimation.h`

rotationMode

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

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

Discussion
Possible values are described in "`Rotation Mode Values`" (page 36). The default 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 34)

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` instances that defines the pacing of the each keyframe segment.

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

Discussion
If the receiver defines *n* keyframes, there must be *n*-1 objects in the `timingFunctions` array. Each timing function describes the pacing of one keyframe to keyframe segment.

Special Considerations

The inherited `timingFunction` value is always ignored.

Availability
Available in iOS 2.0 and later.

Declared In
`CAAnimation.h`

values

An array of objects that provide the keyframe values for the receiver.

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

Discussion
The `values` property is ignored when the `path` (page 34) property is used.

Availability
Available in iOS 2.0 and later.

Declared In
`CAAnimation.h`

Constants

Rotation Mode Values

These constants are used by the rotationMode (page 34) 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 33) 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

Overview

The `CALayer` class is the model class for layer-tree objects. It encapsulates the position, size, and transform of a layer, which defines its coordinate system. It also encapsulates the duration and pacing of a layer and its animations by adopting the `CAMediaTiming` protocol, which defines a layer's time space.

Tasks

Creating a Layer

`+ layer` (page 63)
    Creates and returns an instance of `CALayer`.

`- init` (page 72)
    Returns an initialized `CALayer` object.

`- initWithLayer:` (page 72)
    Override to copy or initialize custom fields of the specified layer.

Accessing the Presentation Layer

`- presentationLayer` (page 76)
: Returns a copy of the layer containing all properties as they were at the start of the current transaction, with any active animations applied.

`- modelLayer` (page 75)
: Returns the model layer of the receiver, if it represents a current presentation layer.

Modifying the Layer Geometry

`frame` (page 53) *property*
: Specifies receiver's frame rectangle in the super-layer's coordinate space.

`bounds` (page 48) *property*
: Specifies the bounds rectangle of the receiver. Animatable.

`position` (page 57) *property*
: Specifies the receiver's position in the superlayer's coordinate system. Animatable.

`zPosition` (page 62) *property*
: Specifies the receiver's position on the z axis. Animatable.

`anchorPointZ` (page 46) *property*
: The Z component of the layer's anchor point. Animatable.

`anchorPoint` (page 46) *property*
: Defines the anchor point of the layer's bounds rectangle. Animatable.

`contentsScale` (page 51) *property*
: The scale factor applied to the layer.

`- affineTransform` (page 66)
: Convenience method for getting the `transform` (page 61) property as an affine transform.

`- setAffineTransform:` (page 79)
: Convenience method for setting the `transform` (page 61) property as an affine transform.

`transform` (page 61) *property*
: Specifies the transform applied to the receiver, relative to the center of its bounds. Animatable.

`sublayerTransform` (page 61) *property*
: Specifies a transform applied to each sublayer when rendering. Animatable.

Providing Layer Content

`contents` (page 49) *property*
: An object that provides the contents of the layer. Animatable.

`contentsRect` (page 50) *property*
: A rectangle, in the unit coordinate space, defining the subrectangle of `contents` (page 49) that the receiver should draw. Animatable.

`contentsCenter` (page 49) *property*
: Specifies the area of the content image that should be scaled. Animatable.

`- display` (page 70)
: Reload the content of this layer.

– `displayLayer:` (page 81) *delegate method*
    Allows the delegate to override the `display` (page 70) implementation.

– `drawInContext:` (page 71)
    Draws the receiver’s content in the specified graphics context.

– `drawLayer:inContext:` (page 82) *delegate method*
    Allows the delegate to override the layer’s `drawInContext:` implementation.

`opaque` (page 56) *property*
    This property affects how the layer creates the content directly.

`edgeAntialiasingMask` (page 52) *property*
    A bitmask defining how the edges of the receiver are rasterized.

– `contentsAreFlipped` (page 67)
    Returns whether the layer content is implicitly flipped when rendered.

`geometryFlipped` (page 53) *property*
    Determines if the geometry of the layer and its sublayers are flipped vertically.

Style Attributes

`contentsGravity` (page 50) *property*
    Determines how the receiver's contents are positioned within its bounds.

`opacity` (page 56) *property*
    Determines the opacity of the receiver. Animatable.

`hidden` (page 54) *property*
    Determines whether the receiver is displayed. Animatable.

`masksToBounds` (page 55) *property*
    Determines if the sublayers are clipped to the receiver’s bounds. Animatable.

`doubleSided` (page 52) *property*
    Determines whether the receiver is displayed when facing away from the viewer. Animatable.

`mask` (page 54) *property*
    An optional layer whose alpha channel is used as a mask to select between the layer's background and the result of compositing the layer's contents with its filtered background.

`cornerRadius` (page 51) *property*
    Specifies a radius used to draw the rounded corners of the receiver’s background. Animatable.

`borderWidth` (page 47) *property*
    Specifies the width of the receiver’s border. Animatable.

`borderColor` (page 47) *property*
    The color of the receiver’s border. Animatable.

`backgroundColor` (page 46) *property*
    Specifies the background color of the receiver. Animatable.

`backgroundFilters` (page 47) *property*
    An optional array of CoreImage filters that are applied to the receiver’s background. Animatable.

`shadowOpacity` (page 58) *property*
    Specifies the opacity of the receiver’s shadow. Animatable.

`shadowRadius` (page 59) *property*
    Specifies the blur radius used to render the receiver’s shadow. Animatable.

`shadowOffset` (page 58) *property*
: Specifies the offset of the receiver’s shadow. Animatable.

`shadowColor` (page 58) *property*
: Specifies the color of the receiver’s shadow. Animatable.

`shadowPath` (page 59) *property*
: Defines the shape of the shadow.

`filters` (page 52) *property*
: An array of CoreImage filters that are applied to the contents of the receiver and its sublayers. Animatable.

`compositingFilter` (page 48) *property*
: A CoreImage filter used to composite the receiver’s contents with the background. Animatable.

`style` (page 60) *property*
: An optional dictionary referenced to find property values that aren't explicitly defined by the receiver.

`minificationFilter` (page 55) *property*
: The filter used when reducing the size of the content.

`minificationFilterBias` (page 55) *property*
: The bias factor used by the minification filter to determine the levels of detail.

`magnificationFilter` (page 54) *property*
: The filter used when increasing the size of the content.

Managing the Layer Hierarchy

`sublayers` (page 60) *property*
: An array containing the receiver's sublayers.

`superlayer` (page 61) *property*
: Specifies receiver's superlayer. (read-only)

– `addSublayer:` (page 66)
: Appends the layer to the receiver’s `sublayers` (page 60) array.

– `removeFromSuperlayer` (page 77)
: Removes the layer from the `sublayers` (page 60) array or `mask` (page 54) property of the receiver’s `superlayer` (page 61).

– `insertSublayer:atIndex:` (page 73)
: Inserts the layer as a sublayer of the receiver at the specified index.

– `insertSublayer:below:` (page 74)
: Inserts the layer into the receiver’s sublayers array, below the specified sublayer.

– `insertSublayer:above:` (page 73)
: Inserts the layer into the receiver’s sublayers array, above the specified sublayer.

– `replaceSublayer:with:` (page 78)
: Replaces the layer in the receiver’s sublayers array with the specified new layer.

Updating Layer Display

– `setNeedsDisplay` (page 79)
: Marks the receiver as needing display before the content is next committed.

`needsDisplayOnBoundsChange` (page 56) *property*
Returns whether the receiver must be redisplayed when the bounds rectangle is updated.

`- displayIfNeeded` (page 71)
Displays the layer if it has been marked as needing display.

`- needsDisplay` (page 75)
Returns whether the layer has been marked as requiring display.

`+ needsDisplayForKey:` (page 64)
Returns whether changes to the specified key requires the layer to be redisplayed.

`- setNeedsDisplayInRect:` (page 80)
Marks the region of the receiver within the specified rectangle as needing display.

Layer Animations

`- addAnimation:forKey:` (page 65)
Add an animation object to the receiver's render tree for the specified key.

`- animationForKey:` (page 66)
Returns the animation added to the receiver with the specified identifier.

`- removeAllAnimations` (page 77)
Remove all animations attached to the receiver.

`- removeAnimationForKey:` (page 77)
Remove the animation attached to the receiver with the specified key.

`- animationKeys` (page 67)
Returns an array containing the keys of all animations currently attached to the receiver.

Managing Layer Resizing and Layout

`- setNeedsLayout` (page 80)
Called when the preferred size of the receiver may have changed.

`name` (page 56) *property*
The name of the receiver.

`- preferredFrameSize` (page 76)
Returns the preferred frame size of the layer in the coordinate space of the superlayer.

`- layoutIfNeeded` (page 74)
Recalculate the receiver's layout, if required.

`- layoutSublayers` (page 75)
Called when the layer requires layout.

`- needsLayout` (page 76)
Returns whether the layer has been marked as requiring layout.

Actions

`actions` (page 45) *property*
A dictionary mapping keys to objects that implement the `CAAction` protocol.

+ `defaultActionForKey:` (page 62)
Returns an object that implements the default action for the specified identifier.

- `actionForKey:` (page 64)
Returns an object that implements the action for the specified identifier.

- `actionForLayer:forKey:` (page 81) *delegate method*
Allows the delegate to customize the action for a layer.

Mapping Between Coordinate and Time Spaces

- `convertPoint:fromLayer:` (page 68)
Converts the point from the specified layer's coordinate system to the receiver's coordinate system.

- `convertPoint:toLayer:` (page 68)
Converts the point from the receiver's coordinate system to the specified layer's coordinate system.

- `convertRect:fromLayer:` (page 69)
Converts the rectangle from the specified layer's coordinate system to the receiver's coordinate system.

- `convertRect:toLayer:` (page 69)
Converts the rectangle from the receiver's coordinate system to the specified layer's coordinate system.

- `convertTime:fromLayer:` (page 70)
Converts the time interval from the specified layer's time space to the receiver's time space.

- `convertTime:toLayer:` (page 70)
Converts the time interval from the receiver's time space to the specified layer's time space

Hit Testing

- `hitTest:` (page 72)
Returns the farthest descendant of the receiver in the layer hierarchy (including itself) that contains a specified point.

- `containsPoint:` (page 67)
Returns whether the receiver contains a specified point.

Rendering

- `renderInContext:` (page 77)
Renders the receiver and its sublayers into the specified context.

`shouldRasterize` (page 59) *property*
A Boolean that indicates whether the layer is rendered as a bitmap before compositing. Animatable

`rasterizationScale` (page 57) *property*
The scale at which to rasterize content, relative to the coordinate space of the layer. Animatable

Scrolling

`visibleRect` (page 62) *property*
Returns the visible region of the receiver, in its own coordinate space. (read-only)

`- scrollPoint:` (page 78)
: Scrolls the receiver's closest ancestor `CAScrollLayer` so that the specified point lies at the origin of the layer.

`- scrollRectToVisible:` (page 79)
: Scrolls the receiver's closest ancestor `CAScrollLayer` the minimum distance needed so that the specified rectangle becomes visible.

Modifying the Delegate

`delegate` (page 51) *property*
: Specifies the receiver's delegate object.

Key-Value Coding Extensions

`- shouldArchiveValueForKey:` (page 80)
: Specifies whether the value of the property for a given key is archived.

`+ defaultValueForKey:` (page 63)
: Specifies the default value of the property with the specified key.

Properties

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

actions

A dictionary mapping keys to objects that implement the `CAAction` protocol.

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

Discussion
The default value is `nil`. See `actionForKey:` (page 64) for a description of the action search pattern.

Availability
Available in iOS 2.0 and later.

See Also
`- actionForKey:` (page 64)
`- actionForLayer:forKey:` (page 81)
`+ defaultActionForKey:` (page 62)
`@property style` (page 60)

Declared In
`CALayer.h`

anchorPoint

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

```
@property CGPoint anchorPoint
```

Discussion
Described in the unit coordinate space. The value of this property is specified in points. Defaults to (0.5, 0.5), the center of the bounds rectangle.

See "Layer Geometry and Transforms" in *Core Animation Programming Guide* for more information on the relationship between the `bounds` (page 48), `anchorPoint` (page 46) and `position` (page 57) properties.

Availability
Available in iOS 2.0 and later.

See Also
`@property position` (page 57)

Declared In
`CALayer.h`

anchorPointZ

The Z component of the layer's anchor point. Animatable.

```
@property CGFloat anchorPointZ
```

Discussion
The `anchorPointZ` value is expressed as a distance along the Z axis. Defaults to `0`.

Availability
Available in iOS 3.0 and later.

See Also
`@property anchorPoint` (page 46)

Declared In
`CALayer.h`

backgroundColor

Specifies the background color of the receiver. Animatable.

```
@property CGColorRef backgroundColor
```

Discussion
The default is `nil`.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

backgroundFilters

An optional array of CoreImage filters that are applied to the receiver's background. Animatable.

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

Discussion
Once an array of filters is set properties should be modified by invoking `setValue:forKeyPath:` using the appropriate key path. This requires that you set the name of the background filter to be modified. For example:

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

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

If the inputs of a background filter are directly modified after the filter is attached to a layer, the behavior is undefined.

Special Considerations

While the `CALayer` class exposes this property, Core Image is not available in iOS. Currently the filters available for this property are undefined.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

borderColor

The color of the receiver's border. Animatable.

```
@property CGColorRef borderColor
```

Discussion
Defaults to opaque black.

Availability
Available in iOS 3.0 and later.

Declared In
`CALayer.h`

borderWidth

Specifies the width of the receiver's border. Animatable.

```
@property CGFloat borderWidth
```

Discussion

The border is drawn inset from the receiver's bounds by `borderWidth`. It is composited above the receiver's `contents` (page 49) and `sublayers` (page 60) and includes the effects of the `cornerRadius` (page 51) property. The default is 0.0.

Availability

Available in iOS 3.0 and later.

Declared In

`CALayer.h`

bounds

Specifies the bounds rectangle of the receiver. Animatable.

```
@property CGRect bounds
```

Discussion

The default is an empty rectangle. The value of this property is specified in points.

See "Layer Geometry and Transforms" in *Core Animation Programming Guide* for more information on the relationship between the `bounds` (page 48), `anchorPoint` (page 46) and `position` (page 57) properties.

Availability

Available in iOS 2.0 and later.

Declared In

`CALayer.h`

compositingFilter

A CoreImage filter used to composite the receiver's contents with the background. Animatable.

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

Discussion

If `nil`, the contents are composited using source-over. The default value is `nil`.

Once a filter is set its properties should be modified by invoking `setValue:forKeyPath:` using the appropriate key path. For example:

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

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

If the inputs of the filter are modified directly after the filter is attached to a layer, the behavior is undefined.

Special Considerations

While the `CALayer` class exposes this property, Core Image is not available in iOS. Currently the filters available for this property are undefined.

Availability

Available in iOS 2.0 and later.

See Also

`@property backgroundFilters` (page 47)

Declared In

`CALayer.h`

contents

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

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

Discussion

A layer can set this property to a `CGImageRef` to display the image as its contents. The default value is `nil`.

Availability

Available in iOS 2.0 and later.

See Also

`@property contentsRect` (page 50)

Declared In

`CALayer.h`

contentsCenter

Specifies the area of the content image that should be scaled. Animatable.

```
@property CGRect contentsCenter
```

Discussion

The rectangle is interpreted after the effects of the `contentsRect` property have been applied to the image.

Defaults to the unit rectangle (0.0,0.0) (1.0,1.0) resulting in the entire image being scaled. If the rectangle extends outside the unit rectangle the result is undefined.

When an image is resized due to its `contentsGravity` (page 50) property, its center part implicitly defines the 3x3 grid that controls how the image is scaled to its drawn size. The center part is stretched in both dimensions; the top and bottom parts are only stretched horizontally; the left and right parts are only stretched vertically; the four corner parts are not stretched at all.

Note: If the width or height of `contentsCenter` is `0`, it is implicitly adjusted to the width or height of a single source pixel centered at that position.

Availability
Available in iOS 3.0 and later.

See Also
- `@property contentsRect` (page 50)
- `@property contentsGravity` (page 50)
- `@property contents` (page 49)

Declared In
`CALayer.h`

contentsGravity

Determines how the receiver's contents are positioned within its bounds.

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

Discussion
The possible values for `contentsGravity` are shown in "`Contents Gravity Values`" (page 84). The default value is `kCAGravityResize` (page 85).

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

contentsRect

A rectangle, in the unit coordinate space, defining the subrectangle of `contents` (page 49) that the receiver should draw. Animatable.

```
@property CGRect contentsRect
```

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

If pixels outside the unit rectangles 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 49)

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 `contentScale` default value is 1.0. In certain restricted cases, the value may set the value to `2.0` on hi-dpi devices.

You can change this value as needed to indicate to Core Animation that the bitmap of the backing layer needs to be bigger or smaller. For example, if you set the contents of the view directly, you can change the value to ensure that layer's bitmap matches the size of the image you are using.

Availability
Available in iOS 4.0 and later.

Declared In
`CALayer.h`

cornerRadius

Specifies a radius used to draw the rounded corners of the receiver's background. Animatable.

```
@property CGFloat cornerRadius
```

Discussion
If the radius is greater than 0 the background is drawn with rounded corners. The default value is 0.0.

Availability
Available in iOS 3.0 and later.

Declared In
`CALayer.h`

delegate

Specifies the receiver's delegate object.

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

Discussion
In iOS, if you want to assign a `UIView` object to this property, you *must* assign the view whose layer this is. Assigning a a superview of the layer's view will cause your application to crash during drawing.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

doubleSided

Determines whether the receiver is displayed when facing away from the viewer. Animatable.

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

Discussion
If `NO`, the layer is hidden when facing away from the viewer. Defaults to `YES`.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

edgeAntialiasingMask

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

```
@property unsigned int edgeAntialiasingMask
```

Discussion
For each of the four edges (left, right, bottom, top) if the corresponding bit is set the edge will be antialiased.

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

The mask values are defined in "Edge Antialiasing Mask" (page 83).

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

filters

An array of CoreImage filters that are applied to the contents of the receiver and its sublayers. Animatable.

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

Discussion
Defaults to `nil`. Filter properties should be modified by calling `setValue:forKeyPath:` on each layer that the filter is attached to. If the inputs of the filter are modified directly after the filter is attached to a layer, the behavior is undefined.

Special Considerations

While the `CALayer` class exposes this property, Core Image is not available in iOS. Currently the filters available for this property are undefined.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

frame

Specifies receiver's frame rectangle in the super-layer's coordinate space.

```
@property CGRect frame
```

Discussion
The `value` of frame is derived from the `bounds` (page 48), `anchorPoint` (page 46) and `position` (page 57) properties. When the `frame` is set, the receiver's `position` (page 57) and the size of the receiver's `bounds` (page 48) are changed to match the new frame rectangle. The value of this property is specified in points.

See "Layer Geometry and Transforms" in *Core Animation Programming Guide* for more information on the relationship between the `bounds` (page 48), `anchorPoint` (page 46) and `position` (page 57) properties.

Note: The `frame` property is not directly animatable. Instead you should animate the appropriate combination of the `bounds` (page 48), `anchorPoint` (page 46) and `position` (page 57) properties to achieve the desired result.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

geometryFlipped

Determines if the geometry of the layer and its sublayers are flipped vertically.

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

Discussion
The value of this property does not effect the rendering of the layer's content, the image specified by contents will display the same regardless of the value of `geometryFlipped`.

Defaults to `NO`.

Availability
Available in iOS 3.0 and later.

Declared In
`CALayer.h`

hidden

Determines whether the receiver is displayed. Animatable.

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

Discussion
The default is `NO`.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

magnificationFilter

The filter used when increasing the size of the content.

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

Discussion
The possible values for `magnificationFilter` are shown in "`Scaling Filters`" (page 86). The default value is `kCAFilterLinear` (page 86).

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

mask

An optional layer whose alpha channel is used as a mask to select between the layer's background and the result of compositing the layer's contents with its filtered background.

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

Discussion
Defaults to `nil`.

Special Considerations

When setting the `mask` to a new layer, the new layer's superlayer must first be set to `nil`, otherwise the behavior is undefined.

Availability
Available in iOS 3.0 and later.

Declared In
`CALayer.h`

masksToBounds

Determines if the sublayers are clipped to the receiver's bounds. Animatable.

```
@property BOOL masksToBounds
```

Discussion
If `YES`, an implicit mask matching the layer bounds is applied to the layer, including the effects of the `cornerRadius` (page 51) property. If `YES` and a `mask` (page 54) property is specified, the two masks are multiplied to get the actual mask values. Defaults to `NO`.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

minificationFilter

The filter used when reducing the size of the content.

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

Discussion
The possible values for `minificationFilter` are shown in "`Scaling Filters`" (page 86). The default value is `kCAFilterLinear` (page 86).

Availability
Available in iOS 2.0 and later.

See Also
  `@property minificationFilterBias` (page 55)

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 55) when it is set to `kCAFilterTrilinear` (page 86).

Defaults to `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. Defaults to `nil`.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

needsDisplayOnBoundsChange

Returns whether the receiver must be redisplayed when the bounds rectangle is updated.

```
@property BOOL needsDisplayOnBoundsChange
```

Discussion
When `YES`, `setNeedsDisplay` (page 79) is automatically invoked when the receiver's `bounds` (page 48) is changed. Default value is `NO`.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

opacity

Determines the opacity of the receiver. Animatable.

```
@property float opacity
```

Discussion
Possible values are between 0.0 (transparent) and 1.0 (opaque). The default is 1.0.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

opaque

This property affects how the layer creates the content directly.

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

Discussion
Defaults to `NO`.

Note that this property has no effect for images provided directly by the developer.

This property only affects what happens if `setNeedsDisplay` is called, and then `display` creates a bitmap for the `drawInContext:` method to draw into. In that case whether the generated bitmap has an alpha channel is defined by the value of this property.

This value has no effect for images provided directly by the developer in the `contents` property.

Availability
Available in iOS 2.0 and later.

Related Sample Code
aurioTouch
GLSprite
SpeakHere

Declared In
`CALayer.h`

position

Specifies the receiver's position in the superlayer's coordinate system. Animatable.

```
@property CGPoint position
```

Discussion
The position is relative to `anchorPoint` (page 46). The value of this property is specified in points. The default is (0.0, 0.0).

See "Layer Geometry and Transforms" in *Core Animation Programming Guide* for more information on the relationship between the `bounds` (page 48), `anchorPoint` (page 46) and `position` (page 57) properties.

Availability
Available in iOS 2.0 and later.

See Also
`@property anchorPoint` (page 46)

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 59) 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

Specifies the color of the receiver's shadow. Animatable.

```
@property CGColorRef shadowColor
```

Discussion
The default is opaque black.

Availability
Available in iOS 3.2 and later.

Declared In
`CALayer.h`

shadowOffset

Specifies the offset of the receiver's shadow. Animatable.

```
@property CGSize shadowOffset
```

Discussion
The default is (0.0,-3.0).

Availability
Available in iOS 3.2 and later.

Declared In
`CALayer.h`

shadowOpacity

Specifies the opacity of the receiver's shadow. Animatable.

```
@property float shadowOpacity
```

Discussion
The default is 0.0.

Availability
Available in iOS 3.2 and later.

Declared In
`CALayer.h`

shadowPath

Defines the shape of the shadow.

```
@property CGPathRef shadowPath
```

Discussion
If the value in this property is non-`nil`, the shadow is created using the specified path instead of the layer's composited alpha channel. The path defines the outline of the shadow. It is filled using the non-zero winding rule and the current shadow color, opacity, and blur radius.

Specifying an explicit path usually improves rendering performance. The default value of this property is `NULL`.

Availability
Available in iOS 3.2 and later.

Declared In
`CALayer.h`

shadowRadius

Specifies the blur radius used to render the receiver's shadow. Animatable.

```
@property CGFloat shadowRadius
```

Discussion
The default value is 3.0.

Availability
Available in iOS 3.2 and later.

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 52) 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 55) and `magnificationFilter` (page 54) 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.

Declared In
`CALayer.h`

style

An optional dictionary referenced to find property values that aren't explicitly defined by the receiver.

```
@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 doesn't define a value for an attribute, the receiver's `defaultValueForKey:` (page 63) method is called. Defaults to `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 receiver's sublayers.

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

Discussion
The layers are listed in back to front order. Defaults to `nil`.

Special Considerations

When setting the `sublayers` property to an array populated with layer objects you must ensure that the layers have had their `superlayer` (page 61) set to `nil`.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

sublayerTransform

Specifies a transform applied to each sublayer when rendering. Animatable.

```
@property CATransform3D sublayerTransform
```

Discussion
This property is typically used as the projection matrix to add perspective and other viewing effects to the receiver. Defaults to the identity transform.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

superlayer

Specifies receiver's superlayer. (read-only)

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

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

transform

Specifies the transform applied to the receiver, relative to the center of its bounds. Animatable.

```
@property CATransform3D transform
```

Discussion
Defaults to the identity transform.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

visibleRect

Returns the visible region of the receiver, 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

Specifies the receiver's position on the z axis. Animatable.

```
@property CGFloat zPosition
```

Discussion
Defaults to 0.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

Class Methods

defaultActionForKey:

Returns an object that implements the default action for the specified identifier.

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

Parameters

aKey
The identifier of the action.

Return Value
Returns the object that provides the action for *aKey*.

Discussion
See `actionForKey:` (page 64) for a description of the action search pattern.

Availability
Available in iOS 2.0 and later.

See Also
- `actionForKey:` (page 64)
- `actionForLayer:forKey:` (page 81)
  `@property actions` (page 45)
  `@property style` (page 60)

Declared In
`CALayer.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
`CALayer.h`

layer

Creates and returns an instance of `CALayer`.

```
+ (id)layer
```

Return Value
The initialized `CALayer` object or `nil` if initialization is not successful.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

needsDisplayForKey:

Returns whether changes to the specified key requires 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 display.

Discussion
Subclasses should 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 will also trigger redisplay.

The default implementation returns `NO`.

Availability
Available in iOS 3.0 and later.

See Also
`+ defaultActionForKey:` (page 62)
`+ defaultValueForKey:` (page 63)

Declared In
`CALayer.h`

Instance Methods

actionForKey:

Returns an object that implements the action for the specified identifier.

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

Parameters

aKey

The identifier of the action.

Return Value
Returns the object that provides the action for *aKey*. The object must implement the `CAAction` protocol.

Discussion
There are three types of actions: property changes, externally-defined events, and layer-defined events. Whenever a layer property is modified, the event with the same name as the property is triggered. External events are defined by the owner of the layer calling `actionForKey:` to lookup the action associated with the identifier and directly messaging the returned object (if non-`nil`.)

The default implementation searches for an action object as follows:

- Return the value `NULL` if the search should not continue.
- If defined, return the object provided by the receiver's delegate method `actionForLayer:forKey:` (page 81).
- Return the object that corresponds to the identifier in the receiver's `actions` (page 45) dictionary property.
- If `nil` is returned their is no action specified for requested *`aKey`*.
- Search the `style` (page 60) dictionary recursively for an actions dictionary that contains the identifier.
- Call the receiver's `defaultActionForKey:` (page 62) method and return the result.

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
`– actionForLayer:forKey:` (page 81)
`@property actions` (page 45)
`+ defaultActionForKey:` (page 62)
`@property style` (page 60)

Declared In
`CALayer.h`

addAnimation:forKey:

Add an animation object to the receiver's render tree for the specified key.

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

Parameters

`anim`
The animation to be added to the render tree. Note that the object is copied by the render tree, not referenced. Any subsequent modifications to the object will not be propagated into the render tree.

`key`
A string that specifies an identifier for the animation. Only one animation per unique key is added to the layer. The special key `kCATransition` (page 83) is automatically used for transition animations. The `nil` pointer is also a valid key.

Discussion
Typically this is implicitly invoked through an action that is an CAAnimation object. If the `duration` property of the animation is zero or negative it is given the default duration, either the current value of the `kCATransactionAnimationDuration` (page 121) transaction property, otherwise .25 seconds

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

addSublayer:

Appends the layer to the receiver's `sublayers` (page 60) array.

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

Parameters

aLayer
The layer to be added to the receiver's `sublayers` (page 60) array.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

affineTransform

Convenience method for getting the `transform` (page 61) property as an affine transform.

```
- (CGAffineTransform)affineTransform
```

Return Value
A `CGAffineTransform` instance that best represents the receiver's `transform` (page 61) property.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

animationForKey:

Returns the animation added to the receiver with the specified identifier.

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

Parameters

key
A string that specifies the identifier of the animation.

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

Discussion
Attempting to modify any properties of the returned object will result in undefined behavior.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

animationKeys

Returns an array containing the keys of all animations currently attached to the receiver.

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

Return Value
An array of NSString objects representing the layer's animations.

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

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 whether the layer content is implicitly flipped when rendered.

```
- (BOOL)contentsAreFlipped
```

Return Value
`YES` if the layer contents are implicitly flipped when rendered.

Discussion
When this method returns `YES` the `CGContextRef` object passed to `drawInContext:` (page 71) by the default `display` (page 70) method will have been y- flipped and rectangles passed to `setNeedsDisplayInRect:` (page 80) will be similarly flipped.

Defaults to `NO`.

Subclasses should not attempt to redefine this method.

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.

Return Value
The point converted to the receiver's coordinate system.

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.

Return Value
The point converted to the coordinate system of *layer*.

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.

Return Value
The rectangle converted to the receiver's coordinate system.

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.

Return Value
The rectangle converted to the coordinate system of *layer*.

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

Reload the content of this layer.

```
- (void)display
```

Discussion
Calls the `drawInContext:` (page 71) method, then updates the receiver's `contents` (page 49) property. You should not call this method directly.

Subclasses can override this method to set the `contents` (page 49) property to an appropriate `CGImageRef`.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

displayIfNeeded

Displays the layer if it has been marked as needing display.

```
- (void)displayIfNeeded
```

Discussion
When this message is received the layer will invoke `display` (page 70) if it has been marked as requiring display.

Availability
Available in iOS 3.0 and later.

See Also
`- needsDisplay` (page 75)

Declared In
`CALayer.h`

drawInContext:

Draws the receiver's content in the specified graphics context.

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

Parameters
ctx
The graphics context in which to draw the content.

Discussion
Default implementation does nothing. The context may be clipped to protect valid layer content. Subclasses that wish to find the actual region to draw can call `CGContextGetClipBoundingBox`. Called by the `display` (page 70) method when the `contents` (page 49) property is being updated.

Subclasses can override this method to draw the receiver's content. When drawing, all coordinates should be specified in the logical coordinate space—that is, measured in points.

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 a 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 `CALayer`.

Availability
Available in iOS 2.0 and later.

See Also
`+ layer` (page 63)

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 method.

Subclasses can optionally copy their instance variables into the new object.

Subclasses should always invoke the superclass implementation

Note: Invoking this method in any other situation will produce undefined behavior. Do not use this method to initialize a new layer with an existing layer's content.

Availability

Available in iOS 2.0 and later.

Declared In

`CALayer.h`

insertSublayer:above:

Inserts the layer into the receiver's sublayers array, above the specified sublayer.

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

Parameters

aLayer
: The layer to be inserted to the receiver's sublayer array.

sublayer
: An existing sublayer in the receiver to insert *aLayer* above.

Special Considerations

If *sublayer* is not in the receiver's sublayers (page 60) array, an exception is raised.

Availability

Available in iOS 2.0 and later.

Declared In

`CALayer.h`

insertSublayer:atIndex:

Inserts the layer as a sublayer of the receiver at the specified index.

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

Parameters

aLayer

The layer to be inserted to the receiver's sublayer array.

index

The index in the receiver at which to insert *aLayer*. This value must not be greater than the count of elements in the sublayer array.

Availability

Available in iOS 2.0 and later.

Declared In

`CALayer.h`

insertSublayer:below:

Inserts the layer into the receiver's sublayers array, below the specified sublayer.

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

Parameters

aLayer

The layer to be inserted to the receiver's sublayer array.

sublayer

An existing sublayer in the receiver to insert *aLayer* after.

Discussion

If *sublayer* is not in the receiver's `sublayers` (page 60) array, an exception is raised.

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 superlayers 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

Called when the layer requires layout.

```
- (void)layoutSublayers
```

Discussion
The default implementation invokes the layout manager method `layoutSublayersOfLayer:`, if a layout manager is specified and it implements that method. Subclasses can override this method to provide their own layout algorithm, which must set the frame of each sublayer.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

modelLayer

Returns the model layer of the receiver, if it represents a current presentation layer.

```
- (id)modelLayer
```

Return Value
A layer instance representing the underlying model layer.

Discussion
The result of calling this method after the transaction that produced the presentation layer has completed is undefined.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

needsDisplay

Returns whether the layer has been marked as requiring display.

```
- (BOOL)needsDisplay
```

Return Value
`YES` if the layer has been marked as requiring display.

Availability
Available in iOS 3.0 and later.

Declared In
`CALayer.h`

needsLayout

Returns whether the layer has been marked as requiring layout.

```
- (BOOL)needsLayout
```

Return Value
`YES` if the layer has been marked as requiring layout.

Availability
Available in iOS 3.0 and later.

See Also
`- setNeedsLayout` (page 80)

Declared In
`CALayer.h`

preferredFrameSize

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

```
- (CGSize)preferredFrameSize
```

Return Value
Returns the receiver's preferred frame size.

Discussion
The default implementation calls the layout manager, if one exists and it implements the `preferredSizeOfLayer:` method. Otherwise, it returns the size of the receiver's `bounds` (page 48) rectangle mapped into coordinate space of the receiver's `superlayer` (page 61).

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

presentationLayer

Returns a copy of the layer containing all properties as they were at the start of the current transaction, with any active animations applied.

```
- (id)presentationLayer
```

Return Value
A layer instance representing the current presentation layer.

Discussion
This method provides a close approximation to the version of the layer that is currently being displayed. The `sublayers` (page 60), `mask` (page 54), and `superlayer` (page 61) properties of the returned layer return the presentation versions of these properties. This pattern carries through to the read-only layer methods. For example, sending a `hitTest:` (page 72) message to the `presentationLayer` will query the presentation values of the layer tree.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

removeAllAnimations

Remove all animations attached to the receiver.

```
- (void)removeAllAnimations
```

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

removeAnimationForKey:

Remove the animation attached to the receiver 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

Removes the layer from the `sublayers` (page 60) array or `mask` (page 54) property of the receiver's `superlayer` (page 61).

```
- (void)removeFromSuperlayer
```

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 that the content is rendered in to.

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 Mac 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 47), `filters` (page 52), `compositingFilter` (page 48), or a `mask` (page 54) values. Future versions of Mac 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 layer in the receiver's sublayers array with the specified new layer.

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

Parameters

oldLayer
The layer to be replaced to the receiver's sublayer array.

newLayer
The layer with which to replace *oldLayer* in the receiver's sublayer array.

Discussion
If the receiver is not the superlayer of *oldLayer* the behavior is undefined.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

scrollPoint:

Scrolls the receiver's closest ancestor `CAScrollLayer` so that the specified point lies at the origin of the layer.

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

Parameters

thePoint
: The point in the receiver to scroll to.

Availability
Available in iOS 2.0 and later.

Declared In
`CAScrollLayer.h`

scrollRectToVisible:

Scrolls the receiver's closest ancestor `CAScrollLayer` the minimum distance needed so that the specified rectangle becomes visible.

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

Parameters

theRect
: The rectangle to be made visible.

Availability
Available in iOS 2.0 and later.

Declared In
`CAScrollLayer.h`

setAffineTransform:

Convenience method for setting the `transform` (page 61) property as an affine transform.

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

Parameters

m
: The affine transform to set as the `transform` (page 61) property.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

setNeedsDisplay

Marks the receiver as needing display before the content is next committed.

```
- (void)setNeedsDisplay
```

Discussion
Calling this method will cause the receiver to recache its content. This will result in the layer receiving a `drawInContext:` (page 71) which may result in the delegate receiving either a `displayLayer:` (page 81) or `drawLayer:inContext:` (page 82) message.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

setNeedsDisplayInRect:

Marks the region of the receiver within the specified rectangle as needing display.

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

Parameters

theRect
: The rectangular region of the receiver to mark as invalid; it should be specified in the coordinate system of the receiver.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

setNeedsLayout

Called when the preferred size of the receiver may have changed.

```
- (void)setNeedsLayout
```

Discussion
This method is typically called when the receiver's sublayers have changed. It marks that the receiver sublayers must update their layout (by invoking `layoutSublayers` (page 75) on the receiver and all its superlayers). If the receiver's layout manager implements the `invalidateLayoutOfLayer:` method it is called.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

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
The default implementation returns `YES`. Called by the object's implementation of `encodeWithCoder:`.

Availability
Available in iOS 4.0 and later.

Declared In
`CALayer.h`

Delegate Methods

actionForLayer:forKey:

Allows the delegate to customize the action for a layer.

```
- (id < CAAction >)actionForLayer:(CALayer *)layer forKey:(NSString *)event
```

Parameters

`layer`
The layer that is the target of the action.

`key`
The identifier of the action.

Return Value
Returns an object implementing the `CAAction` protocol. May return `nil` if the delegate doesn't specify a behavior for `key`.

Discussion
See `actionForKey:` (page 64) for a description of the action search pattern.

Availability
Available in iOS 2.0 and later.

See Also
`- actionForLayer:forKey:` (page 81)
`@property actions` (page 45)
`+ defaultActionForKey:` (page 62)
`@property style` (page 60)

Declared In
`CALayer.h`

displayLayer:

Allows the delegate to override the `display` (page 70) implementation.

```
- (void)displayLayer:(CALayer *)layer
```

Parameters

layer
The layer to display.

Discussion
If defined, called by the default implementation of `display`, in which case it should set the layer's contents property.

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

drawLayer:inContext:

Allows the delegate to override the layer's `drawInContext:` implementation.

```
- (void)drawLayer:(CALayer *)layer inContext:(CGContextRef)ctx
```

Parameters

layer
The layer to draw the content of.

ctx
The graphics context to draw in to.

Discussion
If defined, called by the default implementation of `drawInContext:` (page 71).

Availability
Available in iOS 2.0 and later.

Declared In
`CALayer.h`

Constants

Action Identifiers

These constants are the predefined action identifiers used by `actionForKey:` (page 64), `addAnimation:forKey:` (page 65), `defaultActionForKey:` (page 62), `removeAnimationForKey:` (page 77), `actionForLayer:forKey:` (page 81), and the `CAAction` protocol method `runActionForKey:object:arguments:` (page 133).

```
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 52) 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 50) 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 54) and `minificationFilter` (page 55).

```
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`

Transform

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

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 90)
Creates and returns a new instance of `CAMediaTimingFunction` configured with the predefined timing function specified by *`name`*.

`+ functionWithControlPoints::::` (page 90)
Creates and returns a new instance of `CAMediaTimingFunction` timing function modeled as a cubic Bézier curve using the specified control points.

`- initWithControlPoints::::` (page 91)
Returns an initialized timing function modeled as a cubic Bézier curve using the specified control points.

Accessing the Control Points

`- getControlPointAtIndex:values:` (page 91)
    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 92).

Return Value

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

Availability

Available in iOS 2.0 and later.

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 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 90).

```
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 97) *property*
Specifies the key path the receiver animates.

Property Value Calculation Behavior

`cumulative` (page 96) *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 96) *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 97) *property*
: An optional value function that is applied to interpolated values.

Creating an Animation

`+ animationWithKeyPath:` (page 97)
: Creates and returns an `CAPropertyAnimation` instance for the specified key path.

Properties

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

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.

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 100) *property*
Defines the axes in which the layer may be scrolled.

Scrolling the Layer

`– scrollToPoint:` (page 100)
Changes the origin of the receiver to the specified point.

`– scrollToRect:` (page 100)
Scroll the contents of the receiver to ensure that the rectangle is visible.

2009-09-09 | © 2009 Apple Inc. All Rights Reserved.

Properties

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

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 101). 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 100) 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

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 subpixel-antialiased text by itself, whether into an image or a layer, separately 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 Mac 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 106) *property*
The text to be rendered by the receiver.

Text Visual Properties

`font` (page 104) *property*
    The font used to render the receiver's text.

`fontSize` (page 105) *property*
    The font size used to render the receiver's text. Animatable.

`foregroundColor` (page 105) *property*
    The color used to render the receiver's text. Animatable.

Text Alignment and Truncation

`wrapped` (page 106) *property*
    Determines whether the text is wrapped to fit within the receiver's bounds.

`alignmentMode` (page 104) *property*
    Determines how individual lines of text are horizontally aligned within the receiver's bounds.

`truncationMode` (page 106) *property*
    Determines how the text is truncated to fit within the receiver's bounds.

Properties

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

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 107). Defaults to `kCAAlignmentNatural` (page 108).

Availability
Available in iOS 3.2 and later.

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` (Mac 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 106) property is not an `NSAttributedString`.

Note: If the font property specifies a font size (if it is a `CTFontRef`, a `CGFontRef`, an instance of `NSFont`) the font size is ignored.

Availability
Available in iOS 3.2 and later.

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 `font` property is only used when the `string` (page 106) property is not an `NSAttributedString`.

Note: Implicit animation of this property is only enabled in applications compiled for Mac 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 106) property is not an `NSAttributedString`.

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

Availability
Available in iOS 3.2 and later.

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.

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 107). Defaults to `kCATruncationNone` (page 107).

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 106) property.

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

Constants

`kCATruncationNone`

If the `wrapped` (page 106) 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 104) 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

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 80) 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 111)
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 110) *property*
The number of levels of detail maintained by this layer.

`levelsOfDetailBias` (page 110) *property*
The number of magnified levels of detail for this layer.

Layer Tile Size

`tileSize` (page 111) *property*
The maximum size of each tile used to create the layer's content.

Properties

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

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.

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.

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

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 116) message before modifying the layer tree, and a `commit` (page 116) 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 116)
Begin a new transaction for the current thread.

`+ commit` (page 116)
Commit all changes made during the current transaction.

`+ flush` (page 117)
: Flushes any extant implicit transaction.

Overriding Animation Duration and Timing

`+ animationDuration` (page 115)
: Returns the animation duration used by all animations within this transaction group.

`+ setAnimationDuration:` (page 118)
: Sets the animation duration used by all animations within this transaction group.

`+ animationTimingFunction` (page 115)
: Returns the timing function used for all animations within this transaction group.

`+ setAnimationTimingFunction:` (page 118)
: Sets the timing function used for all animations within this transaction group.

Temporarily Disabling Property Animations

`+ disableActions` (page 117)
: Returns whether actions triggered as a result of property changes made within this transaction group are suppressed.

`+ setDisableActions:` (page 119)
: 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 116)
: Returns the completion block object.

`+ setCompletionBlock:` (page 119)
: Sets the completion block object.

Managing Concurrency

`+ lock` (page 118)
: Attempts to acquire a recursive spin-lock lock, ensuring that returned layer values are valid until unlocked.

`+ unlock` (page 120)
: Relinquishes a previously acquired transaction lock.

Getting and Setting Transaction Properties

`+ setValue:forKey:` (page 120)
: Sets the arbitrary keyed-data for the specified key.

`+ valueForKey:` (page 120)
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
This is a convenience method that returns an `NSNumber` containing the seconds for the `valueForKey:` (page 120) value returned by the `kCATransactionAnimationDuration` (page 121) key.

Availability
Available in iOS 3.0 and later.

See Also
`+ setAnimationDuration:` (page 118)

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 120) value returned by the `kCATransactionAnimationTimingFunction` (page 121) key.

Availability
Available in iOS 3.0 and later.

See Also
`+ setAnimationTimingFunction:` (page 118)

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 116)
`+ flush` (page 117)

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 116)
`+ flush` (page 117)

Declared In
`CATransaction.h`

completionBlock

Returns the completion block object.

```
+ (void)completionBlock
```

Discussion
See `setCompletionBlock:` (page 119) for a description of the role of the completion block object.

Availability
Available in iOS 4.0 and later.

See Also
`+ completionBlock` (page 116)

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 120) value returned by the `kCATransactionDisableActions` (page 121) key.

Availability
Available in iOS 3.0 and later.

See Also
`+ setDisableActions:` (page 119)

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 then 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 116)
`+ commit` (page 116)

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 that data the is read, modified, and set is correctly managed.

Availability
Available in iOS 3.0 and later.

See Also
`+ unlock` (page 120)

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
This is a convenience method that sets an `NSNumber` containing the seconds for the `valueForKey:` (page 120) value of the `kCATransactionAnimationDuration` (page 121) key.

Availability
Available in iOS 3.0 and later.

See Also
`+ animationDuration` (page 115)

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 120) value of the `kCATransactionAnimationTimingFunction` (page 121) key.

Availability
Available in iOS 3.0 and later.

See Also
`+ animationTimingFunction` (page 115)

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 116)

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 120) with an `NSNumber` containing a `YES` for the `kCATransactionDisableActions` (page 121) key.

Availability
Available in iOS 3.0 and later.

See Also
`+ disableActions` (page 117)

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.

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 118)

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 120) and `setValue:forKey:` (page 120).

```
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 (on Mac OS X) by providing a custom `CIFilter` instance.

Tasks

Transition Start and End Point

`startProgress` (page 125) *property*
Indicates the start point of the receiver as a fraction of the entire transition.

`endProgress` (page 124) *property*
Indicates the end point of the receiver as a fraction of the entire transition.

Transition Properties

`type` (page 125) *property*
Specifies the predefined transition type.

`subtype` (page 125) *property*
Specifies an optional subtype that indicates the direction for the predefined motion-based transitions.

Custom Transition Filter

`filter` (page 124) *property*
An optional Core Image filter object that provides the transition.

Properties

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

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 125), and not greater than 1.0. If `endProgress` is less than `startProgress` (page 125) 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` and `kCIInputTargetImageKey` input keys, and the `kCIOutputImageKey` output key. The filter may optionally support the `kCIInputExtentKey` 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 125) and `subtype` (page 125) properties are ignored.

Special Considerations

While the `CATransition` class exposes this property, Core Image is not available in iOS. Currently the filters available for this property are undefined.

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 126). The default is `nil`.

This property is ignored if a custom transition is specified in the `filter` (page 124) 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 126). This property is ignored if a custom transition is specified in the `filter` (page 124) property. The default is `kCATransitionFade` (page 126).

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 125) 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 126) 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 126) 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 126) 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 125) 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`

NSValue Core Animation Additions

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/QuartzCore.framework
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 129)
Creates and returns an NSValue object that contains a given `CATransform3D` structure.

Accessing Data

`– CATransform3DValue` (page 130)
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.

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`

PART II

Protocols

2009-09-09 | © 2009 Apple Inc. All Rights Reserved.

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 133) message.

Tasks

Responding to an Action

– `runActionForKey:object:arguments:` (page 133) *required method*
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`

CALayoutManager Protocol Reference

(informal protocol)

Framework	/System/Library/Frameworks/QuartzCore.framework
Declared in	CALayer.h
Companion guides	Core Animation Programming Guide Core Animation Cookbook

Overview

`CALayoutManager` is an informal protocol implemented by Core Animation layout managers. If a layer's sublayers require custom layout you create a class that implements this protocol and set it as the layer's layout manager using the `CALayer` method `setLayoutManager:`. Your custom layout manager is then used when the layer invokes `setNeedsLayout` (page 80) or `layoutSublayers` (page 75).

Tasks

Layout Layers

`– layoutSublayersOfLayer:` (page 135)
Layout each of the sublayers in the specified layer.

Instance Methods

layoutSublayersOfLayer:

Layout each of the sublayers in the specified layer.

```
- (void)layoutSublayersOfLayer:(CALayer *)layer
```

Parameters

`layer`
The layer that requires layout of its sublayers.

Discussion
This method is called when the sublayers of the *`layer`* may need rearranging, and is typically called when a sublayer has changed its size. The receiver is responsible for changing the frame of each sublayer that requires layout.

Availability
Available in iOS 3.0 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 210) 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 138) *required property*
Specifies the begin time of the receiver in relation to its parent object, if applicable. (required)

`timeOffset` (page 140) *required property*
Specifies an additional time offset in active local time. (required)

Repeating Animations

`repeatCount` (page 139) *required property*
Determines the number of times the animation will repeat. (required)

`repeatDuration` (page 140) *required property*
Determines how many seconds the animation will repeat for. (required)

Duration and Speed

`duration` (page 139) *required property*
Specifies the basic duration of the animation, in seconds. (required)

`speed` (page 140) *required property*
Specifies how time is mapped to receiver's time space from the parent time space. (required)

Playback Modes

`autoreverses` (page 138) *required property*
Determines if the receiver plays in the reverse upon completion. (required)

`fillMode` (page 139) *required property*
Determines if the receiver's presentation is frozen or removed once its active duration has completed. (required)

Properties

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

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.

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 141). The default is `kCAFillModeRemoved` (page 141).

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 140) and `repeatCount` (page 139) 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 140) and `repeatCount` (page 139) 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 139) 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 Mac OS X v10.5 shipped.

  Available in iOS 2.0 and later.

  Declared in `CAMediaTiming.h`.

Declared In

`CAMediaTiming.h`

Other References

2009-09-09 | © 2009 Apple Inc. All Rights Reserved.

Core Video Reference

Framework:	QuartzCore/QuartzCore.h
Declared in	CVBuffer.h CVDisplayLink.h CVImageBuffer.h CVOpenGLBuffer.h CVOpenGLBufferPool.h CVOpenGLTexture.h CVOpenGLTextureCache.h CVPixelBuffer.h CVPixelBufferPool.h CVPixelFormatDescription.h
Companion guide	Core Video Programming Guide

Overview

Core Video is a new pipeline model for digital video in Mac OS X. Partitioning the processing into discrete steps makes it simpler for developers to access and manipulate individual frames without having to worry about translating between data types (QuickTime, OpenGL, and so on) or display synchronization issues.

Core Video is available in:

- Mac OS X v10.4 and later
- Mac OS X v10.3 when QuickTime 7.0 or later is installed
- iOS 4.0 and later

Functions by Task

CVBuffer Functions

Core Video buffer functions operate on all Core Video buffer types, including pixel buffers and OpenGL buffers, as well as OpenGL textures.

`CVBufferGetAttachment` (page 148)
    Returns a specific attachment of a Core Video buffer.

`CVBufferGetAttachments` (page 149)
    Returns all attachments of a Core Video buffer.

`CVBufferPropagateAttachments` (page 150)
: Copies all propagatable attachments from one Core Video buffer to another.

`CVBufferRelease` (page 150)
: Releases a Core Video buffer.

`CVBufferRemoveAllAttachments` (page 151)
: Removes all attachments of a Core Video buffer.

`CVBufferRemoveAttachment` (page 151)
: Removes a specific attachment of a Core Video buffer.

`CVBufferRetain` (page 152)
: Retains a Core Video buffer.

`CVBufferSetAttachment` (page 152)
: Sets or adds an attachment of a Core Video buffer.

`CVBufferSetAttachments` (page 153)
: Sets a set of attachments for a Core Video buffer.

CVHostTime Functions

`CVGetCurrentHostTime` (page 153)
: Retrieves the current value of the host time base.

`CVGetHostClockFrequency` (page 154)
: Retrieve the frequency of the host time base.

`CVGetHostClockMinimumTimeDelta` (page 154)
: Retrieve the smallest possible increment in the host time base.

CVImageBuffer Functions

The functions in this section operate on Core Video buffers derived from the CVImageBuffer abstract type (`CVImageBufferRef`); specifically, pixel buffers, OpenGL buffers, and OpenGL textures.

`CVImageBufferGetCleanRect` (page 154)
: Returns the source rectangle of a Core Video image buffer that represents the clean aperture of the buffer in encoded pixels.

`CVImageBufferGetDisplaySize` (page 155)
: Returns the nominal output display size, in square pixels, of a Core Video image buffer.

`CVImageBufferGetEncodedSize` (page 155)
: Returns the full encoded dimensions of a Core Video image buffer.

CVPixelBuffer Functions

A pixel buffer stores an image in main memory

`CVPixelBufferCreate` (page 156)
: Creates a single pixel buffer for a given size and pixel format.

`CVPixelBufferCreateResolvedAttributesDictionary` (page 157)
: Takes an array of `CFDictionary` objects describing various pixel buffer attributes and tries to resolve them into a single dictionary.

`CVPixelBufferCreateWithBytes` (page 157)
: Creates a pixel buffer for a given size and pixel format containing data specified by a memory location.

`CVPixelBufferCreateWithPlanarBytes` (page 159)
: Creates a single pixel buffer in planar format for a given size and pixel format containing data specified by a memory location.

`CVPixelBufferFillExtendedPixels` (page 160)
: Fills the extended pixels of the pixel buffer.

`CVPixelBufferGetBaseAddress` (page 160)
: Returns the base address of the pixel buffer.

`CVPixelBufferGetBaseAddressOfPlane` (page 161)
: Returns the base address of the plane at the specified plane index.

`CVPixelBufferGetBytesPerRow` (page 161)
: Returns the number of bytes per row of the pixel buffer.

`CVPixelBufferGetBytesPerRowOfPlane` (page 162)
: Returns the number of bytes per row for a plane at the specified index in the pixel buffer.

`CVPixelBufferGetDataSize` (page 162)
: Returns the data size for contiguous planes of the pixel buffer.

`CVPixelBufferGetExtendedPixels` (page 163)
: Returns the amount of extended pixel padding in the pixel buffer.

`CVPixelBufferGetHeight` (page 163)
: Returns the height of the pixel buffer.

`CVPixelBufferGetHeightOfPlane` (page 164)
: Returns the height of the plane at planeIndex in the pixel buffer.

`CVPixelBufferGetPixelFormatType` (page 164)
: Returns the pixel format type of the pixel buffer.

`CVPixelBufferGetPlaneCount` (page 165)
: Returns number of planes of the pixel buffer.

`CVPixelBufferGetTypeID` (page 165)
: Returns the Core Foundation ID of the pixel buffer type.

`CVPixelBufferGetWidth` (page 165)
: Returns the width of the pixel buffer.

`CVPixelBufferGetWidthOfPlane` (page 166)
: Returns the width of the plane at a given index in the pixel buffer.

`CVPixelBufferIsPlanar` (page 166)
: Determine if the pixel buffer is planar.

`CVPixelBufferLockBaseAddress` (page 167)
: Locks the base address of the pixel buffer.

`CVPixelBufferRelease` (page 170)
: Releases a pixel buffer.

`CVPixelBufferRetain` (page 171)
: Retains a pixel buffer.

`CVPixelBufferUnlockBaseAddress` (page 171)
Unlocks the base address of the pixel buffer.

CVPixelBufferPool Functions

`CVPixelBufferPoolCreate` (page 167)
Creates a pixel buffer pool.

`CVPixelBufferPoolCreatePixelBuffer` (page 168)
Creates a pixel buffer from a pixel buffer pool.

`CVPixelBufferPoolGetAttributes` (page 168)
Returns the pool attributes dictionary for a pixel buffer pool.

`CVPixelBufferPoolGetPixelBufferAttributes` (page 169)
Returns the attributes of pixel buffers that will be created from this pool.

`CVPixelBufferPoolGetTypeID` (page 169)
Returns the Core Foundation ID of the pixel buffer pool type.

`CVPixelBufferPoolRelease` (page 169)
Releases a pixel buffer pool.

`CVPixelBufferPoolRetain` (page 170)
Retains a pixel buffer pool.

CVPixelFormatDescription Functions

Used only if you are defining a custom pixel format.

`CVPixelFormatDescriptionRegisterDescriptionWithPixelFormatType` (page 172)
Registers a pixel format description with Core Video.

`CVPixelFormatDescriptionCreateWithPixelFormatType` (page 172)
Creates a pixel format description from a given `OSType` identifier.

`CVPixelFormatDescriptionArrayCreateWithAllPixelFormatTypes` (page 171)
Returns all the pixel format descriptions known to Core Video.

Functions

CVBufferGetAttachment

Returns a specific attachment of a Core Video buffer.

```
CFTypeRef CVBufferGetAttachment (
   CVBufferRef buffer,
   CFStringRef key,
   CVAttachmentMode *attachmentMode
);
```

Parameters

buffer
: The Core Video buffer whose attachment you want to obtain.

key
: A key in the form of a Core Foundation string identifying the desired attachment.

attachmentMode
: On return, `attachmentMode` points to the mode of the attachment. See "CVBuffer Attachment Modes" (page 182) for possible values. If the attachment mode is not defined, this parameter returns `NULL`.

Return Value
If found, the specified attachment.

Discussion
You can attach any Core Foundation object to a Core Video buffer to store additional information by calling `CVBufferSetAttachment` (page 152) or `CVBufferSetAttachments` (page 153).

You can find predefined attachment keys in "CVBuffer Attachment Keys" (page 182) and "Image Buffer Attachment Keys" (page 185).

Availability
Available in iOS 4.0 and later.

Declared In
`CVBuffer.h`

CVBufferGetAttachments

Returns all attachments of a Core Video buffer.

```
CFDictionaryRef CVBufferGetAttachments (
   CVBufferRef buffer,
   CVAttachmentMode attachmentMode
);
```

Parameters

buffer
: The Core Video buffer whose attachments you want to obtain.

attachmentMode
: The mode of the attachments you want to obtain. See "CVBuffer Attachment Modes" (page 182) for possible values.

Return Value
A Core Foundation dictionary with all buffer attachments identified by keys. If no attachment is present, the dictionary is empty. Returns `NULL` for an invalid attachment mode.

Discussion
`CVBufferGetAttachments` is a convenience call that returns all attachments with their corresponding keys in a Core Foundation dictionary.

You can find predefined attachment keys in "CVBuffer Attachment Keys" (page 182) and "Image Buffer Attachment Keys" (page 185).

Availability
Available in iOS 4.0 and later.

Declared In
`CVBuffer.h`

CVBufferPropagateAttachments

Copies all propagatable attachments from one Core Video buffer to another.

```
void CVBufferPropagateAttachments (
   CVBufferRef sourceBuffer,
   CVBufferRef destinationBuffer
);
```

Parameters

sourceBuffer
The buffer to copy attachments from.

destinationBuffer
The buffer to copy attachments to.

Discussion
`CVBufferPropagateAttachments` is a convenience call that copies all attachments with a mode of `kCVAttachmentMode_ShouldPropagate` from one buffer to another.

Availability
Available in iOS 4.0 and later.

Declared In
`CVBuffer.h`

CVBufferRelease

Releases a Core Video buffer.

```
void CVBufferRelease (
   CVBufferRef buffer
);
```

Parameters

buffer
The Core Video buffer that you want to release.

Discussion
Like `CFRelease` `CVBufferRelease` decrements the retain count of a Core Video buffer. If that count consequently becomes zero the memory allocated to the object is deallocated and the object is destroyed. Unlike `CFRelease`, you can pass `NULL` to `CVBufferRelease` without causing a crash.

Availability
Available in iOS 4.0 and later.

Declared In
`CVBuffer.h`

CVBufferRemoveAllAttachments

Removes all attachments of a Core Video buffer.

```
void CVBufferRemoveAllAttachments (
   CVBufferRef buffer
);
```

Parameters

`buffer`
: The Core Video buffer whose attachments you want to remove.

Discussion
`CVBufferRemoveAllAttachments` removes all attachments of a buffer and decrements their reference counts.

Availability
Available in iOS 4.0 and later.

Declared In
`CVBuffer.h`

CVBufferRemoveAttachment

Removes a specific attachment of a Core Video buffer.

```
void CVBufferRemoveAttachment (
   CVBufferRef buffer,
   CFStringRef key
);
```

Parameters

`buffer`
: The Core Video buffer containing the attachment to remove.

`key`
: A key in the form of a Core Foundation string identifying the desired attachment.

Discussion
`CVBufferRemoveAttachment` removes an attachment identified by a key. If found the attachment is removed and the retain count decremented.

You can find predefined attachment keys in "CVBuffer Attachment Keys" (page 182) and "Image Buffer Attachment Keys" (page 185).

Availability
Available in iOS 4.0 and later.

Declared In
`CVBuffer.h`

CVBufferRetain

Retains a Core Video buffer.

```
CVBufferRef CVBufferRetain (
   CVBufferRef buffer
);
```

Parameters

`buffer`
The Core Video buffer that you want to retain.

Return Value
For convenience, the same Core Video buffer you wanted to retain.

Discussion
Like `CFRetain`, `CVBufferRetain` increments the retain count of a Core Video buffer. Unlike `CFRetain`, you can pass `NULL` to `CVBufferRetain` without causing a crash.

Availability
Available in iOS 4.0 and later.

Declared In
`CVBuffer.h`

CVBufferSetAttachment

Sets or adds an attachment of a Core Video buffer.

```
void CVBufferSetAttachment (
   CVBufferRef buffer,
   CFStringRef key,
   CFTypeRef value,
   CVAttachmentMode attachmentMode
);
```

Parameters

`buffer`
The Core Video buffer to which to add or set the attachment.

`key`
The key, in the form of a Core Foundation string, identifying the desired attachment.

`value`
The attachment in the form of a Core Foundation object. If this parameter is `NULL`, the function returns an error.

`attachmentMode`
Specifies the attachment mode for this attachment. See "CVBuffer Attachment Modes" (page 182) for possible values. Any given attachment key may exist in only one mode at a time.

Discussion

You can attach any Core Foundation object to a Core Video buffer to store additional information. If the key doesn't currently exist for the buffer object when you call this function, the new attachment will be added. If the key does exist, the existing attachment will be replaced. In both cases the retain count of the attachment will be incremented. The value can be any CFType. You can find predefined attachment keys in "CVBuffer Attachment Keys" (page 182) and "Image Buffer Attachment Keys" (page 185).

You can also set attachments when creating a buffer by specifying them in the `kCVBufferPropagatedAttachmentsKey` or `kkCVBufferNonpropagatedAttachmentsKey` attributes when creating the buffer.

To retrieve attachments, use the `CVBufferGetAttachment` (page 148) or `CVBufferGetAttachments` (page 149) functions.

Availability

Available in iOS 4.0 and later.

Declared In

`CVBuffer.h`

CVBufferSetAttachments

Sets a set of attachments for a Core Video buffer.

```
void CVBufferSetAttachments (
   CVBufferRef buffer,
   CFDictionaryRef theAttachments,
   CVAttachmentMode attachmentMode
);
```

Parameters

`buffer`

The Core Video buffer to which to set the attachments.

`theAttachments`

The attachments to set, in the form of a Core Foundation dictionary array.

`attachmentMode`

Specifies which attachment mode is desired for this attachment. A particular attachment key may only exist in a single mode at a time.

Discussion

`CVBufferSetAttachments` is a convenience call that in turn calls `CVBufferSetAttachment` (page 152) for each key and value in the given dictionary. All key-value pairs must be in the root level of the dictionary.

Availability

Available in iOS 4.0 and later.

Declared In

`CVBuffer.h`

CVGetCurrentHostTime

Retrieves the current value of the host time base.

```
uint64_t CVGetCurrentHostTime
```

Return Value
The current host time.

Discussion
In Mac OS X, the host time base for CoreVideo and CoreAudio are identical, so the values returned from either API can be used interchangeably.

Availability
Available in iOS 4.0 and later.

Declared In
`CVHostTime.h`

CVGetHostClockFrequency

Retrieve the frequency of the host time base.

```
double CVGetHostClockFrequency
```

Return Value
The current host frequency.

Discussion
In Mac OS X, the host time base for CoreVideo and CoreAudio are identical, and the values returned from either API can be used interchangeably.

Availability
Available in iOS 4.0 and later.

Declared In
`CVHostTime.h`

CVGetHostClockMinimumTimeDelta

Retrieve the smallest possible increment in the host time base.

```
uint32_t CVGetHostClockMinimumTimeDelta
```

Return Value
The smallest valid increment in the host time base.

Availability
Available in iOS 4.0 and later.

Declared In
`CVHostTime.h`

CVImageBufferGetCleanRect

Returns the source rectangle of a Core Video image buffer that represents the clean aperture of the buffer in encoded pixels.

```
CGRect CVImageBufferGetCleanRect (
   CVImageBufferRef imageBuffer
);
```

Parameters

imageBuffer
The image buffer that you want to retrieve the display size from.

Return Value
A `CGRect` structure returning the nominal display size of the buffer. Returns a rectangle of zero size if called with either a non-`CVImageBufferRef` type or `NULL`.

Discussion
The clean aperture size is smaller than the full size of the image. For example, an NTSC DV frame would return a `CGRect` structure with an origin of (8,0) and a size of (704,480). Note that the origin of this rectangle is always in the lower-left corner. This is the same coordinate system as that used by Quartz and Core Image.

Availability
Available in iOS 4.0 and later.

Declared In
`CVImageBuffer.h`

CVImageBufferGetDisplaySize

Returns the nominal output display size, in square pixels, of a Core Video image buffer.

```
CGSize CVImageBufferGetDisplaySize (
   CVImageBufferRef imageBuffer
);
```

Parameters

imageBuffer
The image buffer that you want to retrieve the display size from.

Return Value
A `CGSize` structure defining the nominal display size of the buffer Returns zero size if called with a non-`CVImageBufferRef` type or `NULL`.

Discussion
For example, for an NTSC DV frame this would be 640 x 480.

Availability
Available in iOS 4.0 and later.

Declared In
`CVImageBuffer.h`

CVImageBufferGetEncodedSize

Returns the full encoded dimensions of a Core Video image buffer.

```
CGSize CVImageBufferGetEncodedSize (
   CVImageBufferRef imageBuffer
);
```

Parameters

imageBuffer

The image buffer that you want to retrieve the encoded size from.

Return Value

A `CGSize` structure defining the full encoded size of the buffer. Returns zero size if called with either a non-`CVImageBufferRef` type or `NULL`.

Discussion

For example, for an NTSC DV frame, the encoded size would be 720 x 480. Note: When creating a Core Image image from a Core Video image buffer, you use this call to retrieve the image size.

Availability

Available in iOS 4.0 and later.

Declared In

`CVImageBuffer.h`

CVPixelBufferCreate

Creates a single pixel buffer for a given size and pixel format.

```
CVReturn CVPixelBufferCreate (
   CFAllocatorRef allocator,
   size_t width,
   size_t height,
   OSType pixelFormatType,
   CFDictionaryRef pixelBufferAttributes,
   CVPixelBufferRef *pixelBufferOut
);
```

Parameters

allocator

The allocator to use to create the pixel buffer. Pass `NULL` to specify the default allocator.

width

Width of the pixel buffer, in pixels.

height

Height of the pixel buffer, in pixels.

pixelFormatType

The pixel format identified by its respective four-character code (type `OSType`).

pixelBufferAttributes

A dictionary with additional attributes for a pixel buffer. This parameter is optional. See "Pixel Buffer Attribute Keys" (page 189) for more details.

pixelBufferOut

On return, `pixelBufferOut` points to the newly created pixel buffer.

Return Value

A Core Video result code. See "Result Codes" (page 205) for possible values.

Discussion
This function allocates the necessary memory based on the pixel dimensions, format, and extended pixels described in the pixel buffer's attributes.

Some of the parameters specified in this call override equivalent pixel buffer attributes. For example, if you define the `kCVPixelBufferWidth` and `kCVPixelBufferHeight` keys in the pixel buffer attributes parameter (`pixelBufferAttributes`), these values are overridden by the `width` and `height` parameters.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferCreateResolvedAttributesDictionary

Takes an array of `CFDictionary` objects describing various pixel buffer attributes and tries to resolve them into a single dictionary.

```
CVReturn CVPixelBufferCreateResolvedAttributesDictionary (
   CFAllocatorRef allocator,
   CFArrayRef attributes,
   CFDictionaryRef *resolvedDictionaryOut
);
```

Parameters

`allocator`
  The allocator to use to create the pixel buffer. Pass `NULL` to specify the default allocator.

`attributes`
  An array of Core Foundation dictionaries containing pixel buffer attribute key-value pairs.

`resolvedDictionaryOut`
  On return, `resolvedDictionaryOut` points to the consolidated dictionary.

Return Value
A Core Video result code. See "Result Codes" (page 205) for possible values.

Discussion
This call is useful when you need to resolve requirements between several potential clients of a buffer.

If two or more dictionaries contain the same key but different values, errors may occur. For example, the width and height attributes must match, but if the bytes-per-row (rowBytes) attributes differ, the least common multiple is taken. Mismatches in pixel format allocators or callbacks also cause an error.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferCreateWithBytes

Creates a pixel buffer for a given size and pixel format containing data specified by a memory location.

```
CVReturn CVPixelBufferCreateWithBytes (
   CFAllocatorRef allocator,
   size_t width,
   size_t height,
   OSType pixelFormatType,
   void *baseAddress,
   size_t bytesPerRow,
   CVPixelBufferReleaseBytesCallback releaseCallback,
   void *releaseRefCon,
   CFDictionaryRef pixelBufferAttributes,
   CVPixelBufferRef *pixelBufferOut
);
```

Parameters

allocator
The allocator to use to create this buffer. Pass `NULL` to specify the default allocator.

width
Width of the pixel buffer, in pixels.

height
Height of the pixel buffer, in pixels.

pixelFormatType
Pixel format identified by its respective four character code (type `OSType`).

baseAddress
A pointer to the base address of the memory storing the pixels.

bytesPerRow
Row bytes of the pixel storage memory.

releaseCallback
The callback function to be called when the pixel buffer is destroyed. This callback allows the owner of the pixels to free the memory. See `CVPixelBufferReleaseBytesCallback` (page 174) for more information.

releaseRefCon
User data identifying the pixel buffer. This value is passed to your pixel buffer release callback.

pixelBufferAttributes
A Core Foundation dictionary with additional attributes for a a pixel buffer. This parameter is optional. See "Pixel Buffer Attribute Keys" (page 189) for more details.

pixelBufferOut
On return, `pixelBufferOut` points to the newly created pixel buffer.

Return Value
A Core Video result code. See "Result Codes" (page 205) for possible values.

Discussion
Some of the parameters specified in this call override equivalent pixel buffer attributes. For example, if you define the `kCVPixelBufferWidth` and `kCVPixelBufferHeight` keys in the pixel buffer attributes parameter (`pixelBufferAttributes`), these values are overridden by the `width` and `height` parameters.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferCreateWithPlanarBytes

Creates a single pixel buffer in planar format for a given size and pixel format containing data specified by a memory location.

```
CVReturn CVPixelBufferCreateWithPlanarBytes (
   CFAllocatorRef allocator,
   size_t width,
   size_t height,
   OSType pixelFormatType,
   void *dataPtr,
   size_t dataSize,
   size_t numberOfPlanes,
   void *planeBaseAddress[],
   size_t planeWidth[],
   size_t planeHeight[],
   size_t planeBytesPerRow[],
   CVPixelBufferReleasePlanarBytesCallback releaseCallback,
   void *releaseRefCon,
   CFDictionaryRef pixelBufferAttributes,
   CVPixelBufferRef *pixelBufferOut
);
```

Parameters

allocator
: The allocator to use to create this buffer. Pass `NULL` to specify the default allocator.

width
: Width of the pixel buffer, in pixels.

height
: Height of the pixel buffer, in pixels.

pixelFormatType
: Pixel format identified by its respective four-character code (type `OSType`).

dataPtr
: A pointer to a plane descriptor block if applicable, or `NULL` if not.

dataSize
: The size of the memory if the planes are contiguous, or `NULL` if not.

numberOfPlanes
: The number of planes.

planeBaseAddress
: The array of base addresses for the planes.

planeWidth
: The array of plane widths.

planeHeight
: The array of plane heights.

planeBytesPerRow
: The array of plane bytes-per-row values.

releaseCallback
: The callback function that gets called when the pixel buffer is destroyed. This callback allows the owner of the pixels to free the memory. See `CVPixelBufferReleaseBytesCallback` (page 174) for more information.

releaseRefCon
: A pointer to user data identifying the pixel buffer. This value is passed to your pixel buffer release callback.

pixelBufferAttributes
: A dictionary with additional attributes for a a pixel buffer. This parameter is optional. See "Pixel Buffer Attribute Keys" (page 189) for more details.

pixelBufferOut
: On return, `pixelBufferOut` points to the newly created pixel buffer.

Return Value
A Core Video result code. See "Result Codes" (page 205) for possible values.

Discussion
Some of the parameters specified in this call override equivalent pixel buffer attributes. For example, if you define the `kCVPixelBufferWidth` and `kCVPixelBufferHeight` keys in the pixel buffer attributes parameter (`pixelBufferAttributes`), these values are overridden by the `width` and `height` parameters.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferFillExtendedPixels

Fills the extended pixels of the pixel buffer.

```
CVReturn CVPixelBufferFillExtendedPixels (
   CVPixelBufferRef pixelBuffer
);
```

Parameters

pixelBuffer
: The pixel buffer whose extended pixels you want to fill.

Discussion
This function replicates edge pixels to fill the entire extended region of the image.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetBaseAddress

Returns the base address of the pixel buffer.

```
void * CVPixelBufferGetBaseAddress (
   CVPixelBufferRef pixelBuffer
);
```

Parameters

pixelBuffer

The pixel buffer whose base address you want to obtain.

Return Value

The base address of the pixels. For chunky buffers, this returns a pointer to the pixel at (0,0) in the buffer For planar buffers this returns a pointer to a `PlanarComponentInfo` structure (as defined by QuickTime in `ImageCodec.h`).

Discussion

Retrieving the base address for a pixel buffer requires that the buffer base address be locked via a successful call to `CVPixelBufferLockBaseAddress` (page 167).

Availability

Available in iOS 4.0 and later.

Declared In

`CVPixelBuffer.h`

CVPixelBufferGetBaseAddressOfPlane

Returns the base address of the plane at the specified plane index.

```
void * CVPixelBufferGetBaseAddressOfPlane (
   CVPixelBufferRef pixelBuffer,
   size_t planeIndex
);
```

Parameters

pixelBuffer

The pixel buffer containing the plane whose base address you want to obtain.

planeIndex

The index of the plane.

Return Value

The base address of the plane, or `NULL` for nonplanar pixel buffers.

Discussion

Retrieving the base address for a pixel buffer requires that the buffer base address be locked by a successful call to `CVPixelBufferLockBaseAddress` (page 167).

Availability

Available in iOS 4.0 and later.

Declared In

`CVPixelBuffer.h`

CVPixelBufferGetBytesPerRow

Returns the number of bytes per row of the pixel buffer.

```
size_t CVPixelBufferGetBytesPerRow (
   CVPixelBufferRef pixelBuffer
);
```

Parameters

pixelBuffer
The pixel buffer whose bytes-per-row value you want to obtain.

Return Value
The number of bytes per row of the image data. For planar buffers this function returns a `rowBytes` value such that `bytesPerRow` * `height` covers the entire image including all planes.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetBytesPerRowOfPlane

Returns the number of bytes per row for a plane at the specified index in the pixel buffer.

```
size_t CVPixelBufferGetBytesPerRowOfPlane (
   CVPixelBufferRef pixelBuffer,
   size_t planeIndex
);
```

Parameters

pixelBuffer
The pixel buffer containing the plane.

planeIndex
The index of the plane whose bytes-per-row value you want to obtain.

Return Value
The number of row bytes of the plane, or `NULL` for nonplanar pixel buffers.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetDataSize

Returns the data size for contiguous planes of the pixel buffer.

```
size_t CVPixelBufferGetDataSize (
   CVPixelBufferRef pixelBuffer
);
```

Parameters

pixelBuffer
The pixel buffer whose data size you want to obtain.

Return Value
The data size as specified in the call to `CVPixelBufferCreateWithPlanarBytes` (page 159).

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetExtendedPixels

Returns the amount of extended pixel padding in the pixel buffer.

```
void CVPixelBufferGetExtendedPixels (
   CVPixelBufferRef pixelBuffer,
   size_t *extraColumnsOnLeft,
   size_t *extraColumnsOnRight,
   size_t *extraRowsOnTop,
   size_t *extraRowsOnBottom
);
```

Parameters

pixelBuffer
The pixel buffer whose extended pixel size you want to obtain.

extraColumnsOnLeft
Returns the pixel row padding to the left. May be `NULL`.

extraColumnsOnRight
Returns the pixel row padding to the right. May be `NULL`.

extraRowsOnTop
Returns the pixel row padding to the top. May be `NULL`.

extraRowsOnBottom
The pixel row padding to the bottom. May be `NULL`.

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetHeight

Returns the height of the pixel buffer.

```
size_t CVPixelBufferGetHeight (
   CVPixelBufferRef pixelBuffer
);
```

Parameters

pixelBuffer
The pixel buffer whose height you want to obtain.

Return Value
The buffer height, in pixels.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetHeightOfPlane

Returns the height of the plane at planeIndex in the pixel buffer.

```
size_t CVPixelBufferGetHeightOfPlane (
   CVPixelBufferRef pixelBuffer,
   size_t planeIndex
);
```

Parameters

pixelBuffer
: The pixel buffer whose plane height you want to obtain.

planeIndex
: The index of the plane.

Return Value
The height of the buffer, in pixels, or `0` for nonplanar pixel buffers.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetPixelFormatType

Returns the pixel format type of the pixel buffer.

```
OSType CVPixelBufferGetPixelFormatType (
   CVPixelBufferRef pixelBuffer
);
```

Parameters

pixelBuffer
: The pixel buffer whose format type you want to obtain.

Return Value
A four-character code `OSType` identifier for the pixel format.

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetPlaneCount

Returns number of planes of the pixel buffer.

```
size_t CVPixelBufferGetPlaneCount (
   CVPixelBufferRef pixelBuffer
);
```

Parameters

pixelBuffer
The pixel buffer whose plane count you want to obtain..

Return Value
The number of planes. Returns `0` for nonplanar pixel buffers.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetTypeID

Returns the Core Foundation ID of the pixel buffer type.

```
CFTypeID CVPixelBufferGetTypeID (
   void
);
```

Return Value
The Core Foundation ID for this type.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetWidth

Returns the width of the pixel buffer.

```
size_t CVPixelBufferGetWidth (
   CVPixelBufferRef pixelBuffer
);
```

Parameters

pixelBuffer
The pixel buffer whose width you want to obtain.

Return Value
The width of the buffer, in pixels.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetWidthOfPlane

Returns the width of the plane at a given index in the pixel buffer.

```
size_t CVPixelBufferGetWidthOfPlane (
   CVPixelBufferRef pixelBuffer,
   size_t planeIndex
);
```

Parameters

pixelBuffer
    The pixel buffer whose plane width you want to obtain.

planeIndex
    The index of the plane at which to obtain the width.

Return Value
The width of the plane, in pixels, or `0` for nonplanar pixel buffers.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferIsPlanar

Determine if the pixel buffer is planar.

```
Boolean CVPixelBufferIsPlanar (
   CVPixelBufferRef pixelBuffer
);
```

Parameters

pixelBuffer
    The pixel buffer to check.

Return Value
Returns `true` if the pixel buffer was created using `CVPixelBufferCreateWithPlanarBytes` (page 159), `false` otherwise.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferLockBaseAddress

Locks the base address of the pixel buffer.

```
CVReturn CVPixelBufferLockBaseAddress (
   CVPixelBufferRef pixelBuffer,
   CVOptionFlags lockFlags
);
```

Parameters

pixelBuffer

The pixel buffer whose base address you want to lock.

lockFlags

No options currently defined; pass `0`.

Return Value

A Core Video result code. See "Result Codes" (page 205) for possible values.

Availability

Available in iOS 4.0 and later.

Declared In

`CVPixelBuffer.h`

CVPixelBufferPoolCreate

Creates a pixel buffer pool.

```
CVReturn CVPixelBufferPoolCreate (
   CFAllocatorRef allocator,
   CFDictionaryRef poolAttributes,
   CFDictionaryRef pixelBufferAttributes,
   CVPixelBufferPoolRef *poolOut
);
```

Parameters

allocator

The allocator to use for allocating this buffer pool. Pass `NULL` to specify the default allocator.

poolAttributes

A Core Foundation dictionary containing the attributes for this pixel buffer pool.

pixelBufferAttributes

A Core Foundation dictionary containing the attributes to be used for creating new pixel buffers within the pool.

poolOut

On return, `poolOut` points to the newly created pixel buffer pool.

Return Value

A Core Video result code. See "Result Codes" (page 205) for possible values.

Availability

Available in iOS 4.0 and later.

Declared In

`CVPixelBufferPool.h`

CVPixelBufferPoolCreatePixelBuffer

Creates a pixel buffer from a pixel buffer pool.

```
CVReturn CVPixelBufferPoolCreatePixelBuffer (
   CFAllocatorRef allocator,
   CVPixelBufferPoolRef pixelBufferPool,
   CVPixelBufferRef *pixelBufferOut
);
```

Parameters

`allocator`
: The allocator to use for creating the pixel buffer. Pass `NULL` to specify the default allocator.

`pixelBufferPool`
: The pixel buffer pool for creating the new pixel buffer.

`pixelBufferOut`
: On return, `pixelBufferOut` points to the newly created pixel buffer.

Return Value
A Core Video result code. See "Result Codes" (page 205) for possible values.

Discussion
This function creates a new pixel buffer using the pixel buffer attributes specified during pool creation. This buffer has default attachments as specified in the `pixelBufferAttributes` parameter of `CVPixelBufferPoolCreate` (page 167) (using either the `kCVBufferPropagatedAttachmentsKey` or `kCVBufferNonPropagatedAttachmentsKey` attributes).

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBufferPool.h`

CVPixelBufferPoolGetAttributes

Returns the pool attributes dictionary for a pixel buffer pool.

```
CFDictionaryRef CVPixelBufferPoolGetAttributes (
   CVPixelBufferPoolRef pool
);
```

Parameters

`pool`
: The pixel buffer pool to retrieve the attributes from.

Return Value
A Core Foundation dictionary containing the pool attributes, or `NULL` on failure.

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBufferPool.h`

CVPixelBufferPoolGetPixelBufferAttributes

Returns the attributes of pixel buffers that will be created from this pool.

```
CFDictionaryRef CVPixelBufferPoolGetPixelBufferAttributes (
   CVPixelBufferPoolRef pool
);
```

Parameters

pool
    The pixel buffer pool to retrieve the attributes from.

Return Value

A Core Foundation dictionary containing the pixel buffer attributes, or `NULL` on failure.

Discussion

This function is provided for those cases where you may need to know some information about the buffers that will be created for you .

Availability

Available in iOS 4.0 and later.

Declared In

`CVPixelBufferPool.h`

CVPixelBufferPoolGetTypeID

Returns the Core Foundation ID of the pixel buffer pool type.

```
CFTypeID CVPixelBufferPoolGetTypeID (
   void
);
```

Return Value

The Core Foundation ID for this type.

Availability

Available in iOS 4.0 and later.

Declared In

`CVPixelBufferPool.h`

CVPixelBufferPoolRelease

Releases a pixel buffer pool.

```
void CVPixelBufferPoolRelease (
   CVPixelBufferPoolRef pixelBufferPool
);
```

Parameters

pixelBufferPool
    The pixel buffer pool that you want to release.

Discussion
This function is equivalent to `CFRelease`, but `NULL` safe.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBufferPool.h`

CVPixelBufferPoolRetain

Retains a pixel buffer pool.

```
CVPixelBufferPoolRef CVPixelBufferPoolRetain (
   CVPixelBufferPoolRef pixelBufferPool
);
```

Parameters

`buffer`
The pixel buffer pool that you want to retain.

Return Value
For convenience, the same pixel buffer pool that you wanted to retain.

Discussion
This function is equivalent to `CFRetain`, but `NULL` safe.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBufferPool.h`

CVPixelBufferRelease

Releases a pixel buffer.

```
void CVPixelBufferRelease (
   CVPixelBufferRef texture
);
```

Parameters

`buffer`
The pixel buffer that you want to release.

Discussion
This function is equivalent to `CFRelease`, but `NULL` safe.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferRetain

Retains a pixel buffer.

```
CVPixelBufferRef CVPixelBufferRetain (
   CVPixelBufferRef texture
);
```

Parameters

buffer
: The pixel buffer that you want to retain.

Return Value

For convenience, the same pixel buffer you want to retain.

Discussion

This function is equivalent to `CFRetain`, but `NULL` safe.

Availability

Available in iOS 4.0 and later.

Declared In

`CVPixelBuffer.h`

CVPixelBufferUnlockBaseAddress

Unlocks the base address of the pixel buffer.

```
CVReturn CVPixelBufferUnlockBaseAddress (
   CVPixelBufferRef pixelBuffer,
   CVOptionFlags unlockFlags
);
```

Parameters

pixelBuffer
: The pixel buffer whose base address you want to unlock.

unlockFlags
: No options currently defined; pass `0`.

Return Value

A Core Video result code. See "Result Codes" (page 205) for possible values.

Discussion

Availability

Available in iOS 4.0 and later.

Declared In

`CVPixelBuffer.h`

CVPixelFormatDescriptionArrayCreateWithAllPixelFormatTypes

Returns all the pixel format descriptions known to Core Video.

```
CFArrayRef CVPixelFormatDescriptionArrayCreateWithAllPixelFormatTypes (
   CFAllocatorRef allocator
);
```

Parameters

allocator

The allocator to use when creating the description. Pass `NULL` to specify the default allocator.

Return Value

An array of Core Foundation dictionaries, each containing a pixel format description. See "Pixel Format Description Keys" (page 191) for a list of keys relevant to the format description.

Availability

Available in iOS 4.0 and later.

Declared In

`CVPixelFormatDescription.h`

CVPixelFormatDescriptionCreateWithPixelFormatType

Creates a pixel format description from a given `OSType` identifier.

```
CFDictionaryRef CVPixelFormatDescriptionCreateWithPixelFormatType (
   CFAllocatorRef allocator,
   OSType pixelFormat
);
```

Parameters

allocator

The allocator to use when creating the description. Pass `NULL` to specify the default allocator.

pixelFormat

A four-character code that identifies the pixel format you want to obtain.

Return Value

A Core Foundation dictionary containing the pixel format description. See "Pixel Format Description Keys" (page 191) for a list of keys relevant to the format description.

Availability

Available in iOS 4.0 and later.

Declared In

`CVPixelFormatDescription.h`

CVPixelFormatDescriptionRegisterDescriptionWithPixelFormatType

Registers a pixel format description with Core Video.

```
void CVPixelFormatDescriptionRegisterDescriptionWithPixelFormatType (
   CFDictionaryRef description,
   OSType pixelFormat
);
```

Parameters

description

A Core Foundation dictionary containing the pixel format description. See "Pixel Format Description Keys" (page 191) for a list of required and optional keys.

pixelFormat

The four-character code (type `OSType`) identifier for this pixel format.

Discussion

If you are using a custom pixel format, you must register the format with Core Video using this function. See Technical Q&A 1401: Registering Custom Pixel Formats with QuickTime and Core Video for more details.

Availability

Available in iOS 4.0 and later.

Declared In

`CVPixelFormatDescription.h`

Callbacks

CVFillExtendedPixelsCallBack

Defines a pointer to a custom extended pixel-fill function, which is called whenever the system needs to pad a buffer holding your custom pixel format.

```
typedef Boolean (*CVFillExtendedPixelsCallBack)(
   CVPixelBufferRef pixelBuffer,
   void *refCon
   );
```

Here is how you would declare a custom fill function named `MyExtendedPixelFillFunc`

```
Boolean MyExtendedPixelFillFunc (
   CVPixelBufferRef pixelBuffer,
   void *refCon
   );
```

Parameters

pixelBuffer

The pixel buffer to be padded.

refCon

A pointer to application-defined data. This is the same value you stored in the `CVFillExtendedPixelsCallbackData` (page 176) structure.

Return Value

Return `true` if the padding was successful, `false` otherwise.

Discussion
For more information on implementing a custom extended pixel-fill callback, see Technical Q&A 1440: Implementing a CVFillExtendedPixelsCallback.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelFormatDescription.h`

CVPixelBufferReleaseBytesCallback

Defines a pointer to a pixel buffer release callback function, which is called when a pixel buffer created by `CVPixelBufferCreateWithBytes` (page 157) is released.

```
typedef void (*CVPixelBufferReleaseBytesCallback)(
   void *releaseRefCon,
   const void *baseAddress
   );
```

You would declare a pixel buffer release callback named `MyPixelBufferReleaseCallback` like this:

```
void MyPixelBufferReleaseCallback(
   void *releaseRefCon,
   const void *baseAddress
   );
```

Parameters

releaseRefCon
: A pointer to application-defined data. This pointer is the same as that passed in the `releaseRefCon` parameter of `CVPixelBufferCreateWithBytes` (page 157).

baseAddress
: A pointer to the base address of the memory holding the pixels. This pointer is the same as that passed in the `baseAddress` parameter of `CVPixelBufferCreateWithBytes` (page 157).

Discussion
You use this callback to release the pixels and perform any other cleanup when a pixel buffer is released.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferReleasePlanarBytesCallback

Defines a pointer to a pixel buffer release callback function, which is called when a pixel buffer created by `CVPixelBufferCreateWithPlanarBytes` (page 159) is released.

```
typedef void (*CVPixelBufferReleasePlanarBytesCallback)(
   void *releaseRefCon,
   const void *dataPtr,
   size_t dataSize,
   size_t numberOfPlanes,
   const void *planeAddresses[]
   );
```

You would declare a callback named `MyPixelBufferReleasePlanarBytes` like this:

```
void MyPixelBufferReleasePlanarBytes)(
   void *releaseRefCon,
   const void *dataPtr,
   size_t dataSize,
   size_t numberOfPlanes,
   const void *planeAddresses[]
   );
```

Parameters

releaseRefCon
: A pointer to application-defined data. This pointer is the same as that passed in the `releaseRefCon` parameter of `CVPixelBufferCreateWithPlanarBytes` (page 159).

dataPtr
: A pointer to a plane descriptor block. This is the same pointer you passed to `CVPixelBufferCreateWithPlanarBytes` (page 159) in the `dataPtr` parameter.

dataSize
: The size value you passed to `CVPixelBufferCreateWithPlanarBytes` (page 159) in the `dataSize` parameter.

numberOfPlanes
: The number of planes value you passed to `CVPixelBufferCreateWithPlanarBytes` (page 159) in the `numberOfPlanes` parameter.

planeAddresses
: A pointer to the base plane address you passed to `CVPixelBufferCreateWithPlanarBytes` (page 159) in the `basePlaneAddress` parameter.

Discussion
You use this callback to release the pixels and perform any other cleanup when a pixel buffer is released.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

Data Types

CVBufferRef

Defines the base type for all Core Video buffers.

```
typedef struct __CVBuffer *CVBufferRef;
```

Discussion
CVBuffers represent an abstract type from which all Core Video buffers derive.

Availability
Available in iOS 4.0 and later.

Declared In
`CVBuffer.h`

CVFillExtendedPixelsCallbackData

Holds information describing a custom extended pixel fill algorithm.

```
typedef struct {
    CFIndex version;
    CVFillExtendedPixelsCallBack    fillCallBack;
    void *refCon;
} CVFillExtendedPixelsCallBackData;
```

Fields

`version`
    The version of this fill algorithm.

`fillCallback`
    A pointer to a custom pixel fill function.

`refCon`
    A pointer to application-defined data that is passed to your custom pixel fill function.

Discussion
You must fill out this structure and store it as part of your pixel format description Core Foundation dictionary (key: `kCVPixelFormatFillExtendedPixelsCallback`, type: `CFData`). However, if your custom pixel format never needs the functionality of `CVPixelBufferFillExtendedPixels` (page 160), you don't need to add this key or implement the associated callback.

For more information about defining a custom pixel format, see "Pixel Format Description Keys" (page 191).

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelFormatDescription.h`

CVImageBufferRef

Defines a Core Video image buffer.

```
typedef CVBufferRef CVImageBufferRef;
```

Discussion
An image buffer is an abstract type representing Core Video buffers that hold images. In Core Video, pixel buffers, OpenGL buffers, and OpenGL textures all derive from the image buffer type.

Availability
Available in iOS 4.0 and later.

Declared In
`CVImageBuffer.h`

CVOptionFlags

Define flags to be used for the display link output callback function.

```
typedef uint64_t CVOptionFlags;
```

Discussion
No flags are currently defined.

Availability
Available in iOS 4.0 and later.

Declared In
`CVBase.h`

CVPixelBufferRef

Defines a Core Video pixel buffer.

```
typedef CVImageBufferRef CVPixelBufferRef;
```

Discussion
The pixel buffer stores an image in main memory.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferPoolRef

Defines a pixel buffer pool.

```
typedef struct _CVPixelBufferPool *CVPixelBufferPoolRef;
```

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBufferPool.h`

CVReturn

Defines the return error code for Core Video functions.

```
typedef int32_t CVReturn;
```

Discussion
See "Result Codes" (page 205) for possible values.

Availability
Available in iOS 4.0 and later.

Declared In
`CVReturn.h`

CVSMPTETime

A structure for holding a SMPTE time.

```
struct CVSMPTETime    {
    SInt16  subframes;
    SInt16  subframeDivisor;
    UInt32  counter;
    UInt32  type;
    UInt32  flags;
    SInt16  hours;
    SInt16  minutes;
    SInt16  seconds;
    SInt16  frames;
    ;}
typedef struct CVSMPTETime CVSMPTETime;
```

Fields

`subframes`
: The number of subframes in the full message.

`subframeDivisor`
: The number of subframes per frame (typically, 80).

`counter`
: The total number of messages received.

`type`
: The kind of SMPTE time type. See "SMPTE Time Types" (page 195) for a list of possible values.

`flags`
: A set of flags that indicate the SMPTE state. See "SMPTE State Flags" (page 195) for possible values.

`hours`
: The number of hours in the full message.

`minutes`
: The number of minutes in the full message.

`seconds`
: The number of seconds in the full message.

`frames`
: The number of frames in the full message.

Availability
Available in iOS 4.0 and later.

Declared In
`CVBase.h`

CVTime

A structure for reporting Core Video time values.

```
typedef struct {
    int64_t     timeValue;
    int64_t     timeScale;
    int32_t     flags;
} CVTime;
```

Fields

`timeValue`
The time value.

`timeScale`
The time scale for this value.

`flags`
Flags associated with the `CVTime` value. See "CVTime Constants" (page 183) for possible values. If `kCVTimeIsIndefinite` is set, you should not use any of the other fields in this structure.

Discussion
This structure is equivalent to the QuickTime `QTTime` structure.

Availability
Available in iOS 4.0 and later.

Declared In
`CVBase.h`

CVTimeStamp

A structure for defining a display timestamp.

```
typedef struct {
    uint32_t    version;
    int32_t     videoTimeScale;
    int64_t     videoTime;
    uint64_t    hostTime;
    double      rateScalar;
    int64_t     videoRefreshPeriod;
    CVSMPTETime smpteTime;
    uint64_t    flags;
    uint64_t    reserved;
} CVTimeStamp;
```

Fields

`version`
The current `CVTimeStamp` structure is version 0. Some functions require you to specify a version when passing in a timestamp structure to be filled.

`videoTimeScale`
The scale (in units per second) of the `videoTimeScale` and `videoRefreshPeriod` fields.

`videoTime`
: The start of a frame (or field for interlaced video).

`hostTime`
: The host root time base time.

`rateScalar`
: The current rate of the device as measured by the timestamps, divided by the nominal rate

`videoPeriod`
: The nominal update period of the current output device.

`smpteTime`
: The SMPTE time representation of the timestamp.

`flags`
: A bit field containing additional information about the timestamp. See "CVTimeStamp Flags" (page 184) for a list of possible values. .

`reserved`
: Reserved. Do not use.

Discussion
This structure is designed to be very similar to the audio time stamp defined in the Core Audio framework. However, unlike the audio timestamps, floating-point values are not used to represent the video equivalent of sample times. This was done partly to avoid precision issues, and partly because QuickTime still uses integers for time values and time scales. In the actual implementation it has turned out to be very convenient to use integers, and we can represent frame rates like NTSC (30000/1001 fps) exactly. The `mHostTime` structure field uses the same Mach absolute time base used in Core Audio, so that clients of the Core Video API can synchronize between the two subsystems.

Availability
Available in iOS 4.0 and later.

Declared In
`CVBase.h`

CVPlanarComponentInfo

A structure for describing planar components.

```
struct CVPlanarComponentInfo {
  int32_t             offset;
  uint32_t            rowBytes;
};
typedef struct CVPlanarComponentInfo CVPlanarComponentInfo;
```

Fields
`offset`
: The offset from the main base address to the base address of this plane. (Big-endian.)

`rowBytes`
: The number of bytes per row of this plane. (Big-endian.)

Discussion
Planar pixel buffers have this descriptor at their base address. Clients should generally use `CVPixelBufferGetBaseAddressOfPlane` (page 161) and `CVPixelBufferGetBytesPerRowOfPlane` (page 162) instead of accessing it directly.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPlanarPixelBufferInfo

A structure for describing planar buffers.

```
struct CVPlanarPixelBufferInfo {
  CVPlanarComponentInfo  componentInfo[1];
};
typedef struct CVPlanarPixelBufferInfo CVPlanarPixelBufferInfo;
```

Fields

`componentInfo`
    Description forthcoming.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

CVPlanarPixelBufferInfo_YCbCrPlanar

A structure for describing YCbCr planar buffers.

```
struct CVPlanarPixelBufferInfo_YCbCrPlanar {
  CVPlanarComponentInfo  componentInfoY;
  CVPlanarComponentInfo  componentInfoCb;
  CVPlanarComponentInfo  componentInfoCr;
};
typedef struct CVPlanarPixelBufferInfo_YCbCrPlanar
CVPlanarPixelBufferInfo_YCbCrPlanar;
```

Fields

`componentInfoY`
    Description forthcoming.

`componentInfoCb`
    Description forthcoming.

`componentInfoCr`
    Description forthcoming.

Availability
Available in iOS 4.0 and later.

Declared In
`CVPixelBuffer.h`

Constants

CVBuffer Attachment Keys

Specify an attachment type for a Core Video buffer.

```
const CFStringRef kCVBufferMovieTimeKey;
const CFStringRef kCVBufferTimeValueKey;
const CFStringRef kCVBufferTimeScaleKey;
```

Constants

`kCVBufferMovieTimeKey`

The movie time associated with the buffer. Generally only available for frames emitted by QuickTime (type `CFDictionary` containing the `kCVBufferTimeValueKey` and `kCVBufferTimeScaleKey` keys).

Available in iOS 4.0 and later.

Declared in `CVBuffer.h`.

`kCVBufferTimeValueKey`

The actual time value associated with the movie.

Available in iOS 4.0 and later.

Declared in `CVBuffer.h`.

`kCVBufferTimeScaleKey`

The time scale associated with the movie.

Available in iOS 4.0 and later.

Declared in `CVBuffer.h`.

CVBuffer Attachment Modes

Specify the propagation mode of a Core Video buffer attachment.

```
enum {
    kCVAttachmentMode_ShouldNotPropagate    = 0,
    kCVAttachmentMode_ShouldPropagate       = 1,
};
typedef uint32_t CVAttachmentMode;
```

Constants

`kCVAttachmentMode_ShouldNotPropagate`

Do not propagate this attachment.

Available in iOS 4.0 and later.

Declared in `CVBuffer.h`.

`kCVAttachmentMode_ShouldPropagate`

Copy this attachment when using the `CVBufferPropagateAttachments` (page 150) function. For example, in most cases, you would want to propagate an attachment bearing a timestamp to each successive buffer.

Available in iOS 4.0 and later.

Declared in `CVBuffer.h`.

Discussion
You set these attributes when adding attachments to a `CVBuffer` object.

CVBuffer Attribute Keys

Specify attributes associated with Core Video buffers.

```
const CFStringRef kCVBufferPropagatedAttachmentsKey;
const CFStringRef kCVBufferNonPropagatedAttachmentsKey;
```

Constants
`kCVBufferPropagatedAttachmentsKey`
Attachments that should be copied when using the `CVBufferPropagateAttachments` (page 150) function (type `CFDictionary`, containing a list of attachments as key-value pairs).

Available in iOS 4.0 and later.

Declared in `CVBuffer.h`.

`kCVBufferNonPropagatedAttachmentsKey`
Attachments that should not be copied when using the `CVBufferPropagateAttachments` (page 150) function (type `CFDictionary`, containing a list of attachments as key-value pairs).

Available in iOS 4.0 and later.

Declared in `CVBuffer.h`.

Discussion
These attributes let you set multiple attachments at the time of buffer creation, rather than having to call `CVBufferSetAttachment` (page 152) for each attachment.

CVTime Constants

Specify flags for the `CVTime` structure.

```
enum {
kCVTimeIsIndefinite = 1 << 0
};
```

Constants
`kCVTimeIsIndefinite`
The time value is unknown.

Available in iOS 4.0 and later.

Declared in `CVBase.h`.

CVTime Values

Indicate specific `CVTime` values.

```
const CVTime kCVZeroTime;
const CVTime kCVIndefiniteTime;
```

Constants

`kCVZeroTime`

Zero time or duration. For example, `CVDisplayLinkGetOutputVideoLatency` returns `kCVZeroTime` for zero video latency.

Available in iOS 4.0 and later.

Declared in `CVBase.h`.

`kCVIndefiniteTime`

An unknown or indefinite time. For example, `CVDisplayLinkGetNominalOutputVideoRefreshPeriod` returns `kCVIndefiniteTime` if the display link specified is not valid.

Available in iOS 4.0 and later.

Declared in `CVBase.h`.

CVTimeStamp Flags

Specify flags for the `CVTimeStamp` structure.

```
enum
{
    kCVTimeStampVideoTimeValid              = (1L << 0),
    kCVTimeStampHostTimeValid               = (1L << 1),
    kCVTimeStampSMPTETimeValid              = (1L << 2),
    kCVTimeStampVideoRefreshPeriodValid     = (1L << 3),
    kCVTimeStampRateScalarValid             = (1L << 4),
    kCVTimeStampTopField                    = (1L << 16),
    kCVTimeStampBottomField                 = (1L << 17)
};
enum
{
    kCVTimeStampVideoHostTimeValid   =
            (kCVTimeStampVideoTimeValid | kCVTimeStampHostTimeValid),
    kCVTimeStampIsInterlaced            =
            (kCVTimeStampTopField | kCVTimeStampBottomField)
};
```

Constants

`kCVTimeStampVideoTimeValid`

The value in the video time field is valid.

Available in iOS 4.0 and later.

Declared in `CVBase.h`.

`kCVTimeStampHostTimeValid`

The value in the host time field is valid.

Available in iOS 4.0 and later.

Declared in `CVBase.h`.

`kCVTimeStampSMPTETimeValid`
: The value in the SMPTE time field is valid.

  Available in iOS 4.0 and later.

  Declared in `CVBase.h`.

`kCVTimeStampVideoRefreshPeriodValid`
: The value in the video refresh period field is valid.

  Available in iOS 4.0 and later.

  Declared in `CVBase.h`.

`kCVTimeStampRateScalarValid`
: The value in the rate scalar field is valid.

  Available in iOS 4.0 and later.

  Declared in `CVBase.h`.

`kCVTimeStampTopField`
: The timestamp represents the top lines of an interlaced image.

  Available in iOS 4.0 and later.

  Declared in `CVBase.h`.

`kCVTimeStampBottomField`
: The timestamp represents the bottom lines of an interlaced image.

  Available in iOS 4.0 and later.

  Declared in `CVBase.h`.

`kCVTimeStampVideoHostTimeValid`
: A convenience constant indicating that both the video time and host time fields are valid.

  Available in iOS 4.0 and later.

  Declared in `CVBase.h`.

`kCVTimeStampIsInterlaced`
: A convenience constant indicating that the timestamp is for an interlaced image.

  Available in iOS 4.0 and later.

  Declared in `CVBase.h`.

Discussion
These flags indicate which fields in the `CVTimeStamp` (page 179) structure contain valid information.

Image Buffer Attachment Keys

Specify attachment types associated with image buffers.

```
const CFStringRef   kCVImageBufferCGColorSpaceKey;
const CFStringRef   kCVImageBufferGammaLevelKey;
const CFStringRef   kCVImageBufferCleanApertureKey;
const CFStringRef   kCVImageBufferPreferredCleanApertureKey;
const CFStringRef   kCVImageBufferCleanApertureWidthKey;
const CFStringRef   kCVImageBufferCleanApertureHeightKey;
const CFStringRef   kCVImageBufferCleanApertureHorizontalOffsetKey;
const CFStringRef   kCVImageBufferCleanApertureVerticalOffsetKey;
const CFStringRef   kCVImageBufferFieldCountKey;
const CFStringRef   kCVImageBufferFieldDetailKey;
const CFStringRef   kCVImageBufferFieldDetailTemporalTopFirst;
const CFStringRef   kCVImageBufferFieldDetailTemporalBottomFirst;
const CFStringRef   kCVImageBufferFieldDetailSpatialFirstLineEarly;
const CFStringRef   kCVImageBufferFieldDetailSpatialFirstLineLate;
const CFStringRef   kCVImageBufferPixelAspectRatioKey;
const CFStringRef   kCVImageBufferPixelAspectRatioHorizontalSpacingKey;
const CFStringRef   kCVImageBufferPixelAspectRatioVerticalSpacingKey;
const CFStringRef   kCVImageBufferDisplayDimensionsKey;
const CFStringRef   kCVImageBufferDisplayWidthKey;
const CFStringRef   kCVImageBufferDisplayHeightKey;
const CFStringRef   kCVImageBufferYCbCrMatrixKey;
const CFStringRef   kCVImageBufferYCbCrMatrix_ITU_R_709_2;
const CFStringRef   kCVImageBufferYCbCrMatrix_ITU_R_601_4;
const CFStringRef   kCVImageBufferYCbCrMatrix_SMPTE_240M_1995;
```

Constants

`kCVImageBufferCGColorSpaceKey`

The color space for the buffer (type `CGColorSpaceRef`).

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferGammaLevelKey`

The gamma level for this buffer (type `CFNumber`).

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferCleanApertureKey`

The clean aperture for the buffer (type `CFDictionary` , containing the clean aperture width, height, and horizontal and vertical offset key-value pairs).

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferPreferredCleanApertureKey`

The preferred clean aperture for the buffer (type `CFDictionary` , containing the clean aperture width, height, and horizontal and vertical offset key-value pairs).

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferCleanApertureWidthKey`

The clean aperture width (type `CFNumber`).

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferCleanApertureHeightKey`
The clean aperture height (type `CFNumber`).

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferCleanApertureHorizontalOffsetKey`
The clean aperture horizontal offset (type `CFNumber`).

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferCleanApertureVerticalOffsetKey`
The clean aperture vertical offset (type `CFNumber`).

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferFieldCountKey`
The field count for the buffer (type `CFNumber`).

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferFieldDetailKey`
Specific information about the field of a video frame in the buffer (type `CFDictionary`, containing the temporal bottom first and top first and spacial first-line-early and first-line-late keys).

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferFieldDetailTemporalTopFirst`
(type `CFString`).

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferFieldDetailTemporalBottomFirst`
(type `CFString`).

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferFieldDetailSpatialFirstLineEarly`
(type `CFString`).

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferFieldDetailSpatialFirstLineLate`
(type `CFString`).

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferPixelAspectRatioKey`
The pixel aspect ratio of the buffer (type `CFDictionary`, containing the horizontal and vertical spacing keys).

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferPixelAspectRatioHorizontalSpacingKey`
: The horizontal component of the buffer aspect ratio (type `CFNumber`).

  Available in iOS 4.0 and later.

  Declared in `CVImageBuffer.h`.

`kCVImageBufferPixelAspectRatioVerticalSpacingKey`
: The vertical component of the buffer aspect ratio (type `CFNumber`).

  Available in iOS 4.0 and later.

  Declared in `CVImageBuffer.h`.

`kCVImageBufferDisplayDimensionsKey`
: The buffer display dimensions (type `CFDictionary` containing the buffer display width and height keys).

  Available in iOS 4.0 and later.

  Declared in `CVImageBuffer.h`.

`kCVImageBufferDisplayWidthKey`
: The buffer display width (type `CFNumber`).

  Available in iOS 4.0 and later.

  Declared in `CVImageBuffer.h`.

`kCVImageBufferDisplayHeightKey`
: The buffer display height (type `CFNumber`).

  Available in iOS 4.0 and later.

  Declared in `CVImageBuffer.h`.

`kCVImageBufferYCbCrMatrixKey`
: The type of conversion matrix used for this buffer when converting from YCbCr to RGB images (type `CFString`). The value for this key should be one of the following constants: `kCVImageBufferYCbCrMatrix_ITU_R_709_2`, `kCVImageBufferYCbCrMatrix_ITU_R_601_4`, or `kCVImageBufferYCbCrMatrix_SMPTE_240M_1995`.

  Available in iOS 4.0 and later.

  Declared in `CVImageBuffer.h`.

`kCVImageBufferYCbCrMatrix_ITU_R_709_2`
: Specifies the YCbCr to RGB conversion matrix for HDTV digital television (ITU R 709) images.

  Available in iOS 4.0 and later.

  Declared in `CVImageBuffer.h`.

`kCVImageBufferYCbCrMatrix_ITU_R_601_4`
: Specifies the YCbCr to RGB conversion matrix for standard digital television (ITU R 601) images.

  Available in iOS 4.0 and later.

  Declared in `CVImageBuffer.h`.

`kCVImageBufferYCbCrMatrix_SMPTE_240M_1995`
: Specifies the YCbCR to RGB conversion matrix for 1920 x 1135 HDTV (SMPTE 240M 1995).

  Available in iOS 4.0 and later.

  Declared in `CVImageBuffer.h`.

Discussion

Image buffer attachment keys are stored in a Core Foundation dictionary associated with an image buffer. Note that some of these keys are stored in subdictionaries keyed by a higher-level attribute. For example, the `kCVImageBufferDisplayWidthKey` and `kCVImageBufferDisplayHeightKey` attributes are stored in a Core Foundation dictionary keyed to the `kCVImageBufferDisplayDimensionsKey` attribute.

Pixel Buffer Attribute Keys

Specify attributes associated with a pixel buffer.

```
const CFStringRef kCVPixelBufferPixelFormatTypeKey;
 const CFStringRef kCVPixelBufferMemoryAllocatorKey;
 const CFStringRef kCVPixelBufferWidthKey;
 const CFStringRef kCVPixelBufferHeightKey;
 const CFStringRef kCVPixelBufferExtendedPixelsLeftKey;
 const CFStringRef kCVPixelBufferExtendedPixelsTopKey;
 const CFStringRef kCVPixelBufferExtendedPixelsRightKey;
 const CFStringRef kCVPixelBufferExtendedPixelsBottomKey;
 const CFStringRef kCVPixelBufferBytesPerRowAlignmentKey;
 const CFStringRef kCVPixelBufferCGBitmapContextCompatibilityKey;
 const CFStringRef kCVPixelBufferCGImageCompatibilityKey;
 const CFStringRef kCVPixelBufferOpenGLCompatibilityKey;
 const CFStringRef kCVPixelBufferIOSurfacePropertiesKey;
 const CFStringRef kCVPixelBufferPlaneAlignmentKey;
```

Constants

`kCVPixelBufferPixelFormatTypeKey`

The pixel format for this buffer (type `CFNumber`, or type `CFArray` containing an array of `CFNumber` types (actually type `OSType`)). For a listing of common pixel formats, see the QuickTime Ice Floe Dispatch 20.

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferMemoryAllocatorKey`

The allocator used with this buffer (type `CFAllocatorRef`).

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferWidthKey`

The width of the pixel buffer (type `CFNumber`).

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferHeightKey`

The height of the pixel buffer (type `CFNumber`).

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferExtendedPixelsLeftKey`

The number of pixels padding the left of the image (type `CFNumber`).

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferExtendedPixelsTopKey`
The number of pixels padding the top of the image (type `CFNumber`).

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferExtendedPixelsRightKey`
The number of pixels padding the right of the image (type `CFNumber`).

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferExtendedPixelsBottomKey`
The number of pixels padding the bottom of the image (type `CFNumber`).

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferBytesPerRowAlignmentKey`
Indicates the number of bytes per row in the pixel buffer (type `CFNumber`).

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferCGBitmapContextCompatibilityKey`
Indicates whether the pixel buffer is compatible with Core Graphics bitmap contexts (type `CFBoolean`).

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferCGImageCompatibilityKey`
Indicates whether the pixel buffer is compatible with `CGImage` types (type `CFBoolean`).

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferOpenGLCompatibilityKey`
Indicates whether the pixel buffer is compatible with OpenGL contexts (type `CFBoolean`).

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferIOSurfacePropertiesKey`
Description forthcoming (type `CFDictionary).`

Presence of this key requests allocation via the IOSurface framework.

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferPlaneAlignmentKey`
Description forthcoming (type `CFNumber`).

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

Discussion
You specify these keys in a Core Foundation dictionary when calling functions such as `CVPixelBufferCreate` (page 156).

Pixel Buffer Pool Attribute Keys

Specify attributes associated with a pixel buffer pool.

```
const CFStringRef kCVPixelBufferPoolMinimumBufferCountKey;
const CFStringRef kCVPixelBufferPoolMaximumBufferAgeKey;
```

Constants

`kCVPixelBufferPoolMinimumBufferCountKey`
: The minimum number of buffers allowed in the pixel buffer pool (type `CFNumber`).

  Available in iOS 4.0 and later.

  Declared in `CVPixelBufferPool.h`.

`kCVPixelBufferPoolMaximumBufferAgeKey`
: The maximum allowable age for a buffer in the pixel buffer pool (type `CFAbsoluteTime`).

  Available in iOS 4.0 and later.

  Declared in `CVPixelBufferPool.h`.

Discussion

You specify these keys in a Core Foundation dictionary when calling functions such as `CVPixelBufferPoolCreate` (page 167).

Pixel Format Description Keys

Specify attributes of a pixel format.

```
const CFStringRef kCVPixelFormatName;
const CFStringRef kCVPixelFormatConstant;
const CFStringRef kCVPixelFormatCodecType;
const CFStringRef kCVPixelFormatFourCC;
const CFStringRef kCVPixelFormatPlanes;
const CFStringRef kCVPixelFormatBlockWidth;
const CFStringRef kCVPixelFormatBlockHeight;
const CFStringRef kCVPixelFormatBitsPerBlock;
const CFStringRef kCVPixelFormatBlockHorizontalAlignment;
const CFStringRef kCVPixelFormatBlockVerticalAlignment;
const CFStringRef kCVPixelFormatBlackBlock;
const CFStringRef kCVPixelFormatHorizontalSubsampling;
const CFStringRef kCVPixelFormatVerticalSubsampling;

const CFStringRef kCVPixelFormatOpenGLFormat;
const CFStringRef kCVPixelFormatOpenGLType;
const CFStringRef kCVPixelFormatOpenGLInternalFormat;

const CFStringRef kCVPixelFormatCGBitmapInfo;

const CFStringRef kCVPixelFormatQDCompatibility;
const CFStringRef kCVPixelFormatCGBitmapContextCompatibility;
const CFStringRef kCVPixelFormatCGImageCompatibility;
const CFStringRef kCVPixelFormatOpenGLCompatibility;

const CFStringRef kCVPixelFormatFillExtendedPixelsCallback;
const CFStringRef kCVPixelFormatDirect3DFormat;
const CFStringRef kCVPixelFormatDirect3DType;
const CFStringRef kCVPixelFormatDirect3DInternalFormat;
const CFStringRef kCVPixelFormatDirect3DCompatibility;
```

Constants

`kCVPixelFormatName`

The name of the pixel format (type `CFString`). This should be the same as the codec name you would use in QuickTime.

Available in iOS 4.0 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatConstant`

The pixel format constant for QuickTime.

Available in iOS 4.0 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatCodecType`

The codec type (type `CFString`). For example, `'2vuy'` or `k422YpCbCr8CodecType`.

Available in iOS 4.0 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatFourCC`

The Microsoft FourCC equivalent code for this pixel format (type `CFString`).

Available in iOS 4.0 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatPlanes`

The number of image planes associated with this format (type `CFNumber`. Each plane may contain a single component or an interleaved set of components. Note that if your pixel format is not planar, you can put the required format keys at the top-level dictionary.

Available in iOS 4.0 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatBlockWidth`

The width, in pixels, of the smallest byte-addressable group of pixels (type `CFNumber`. Used to assist with allocating memory for pixel formats that don't have an integer value for bytes per pixel. Assumed to be 1 if this key is not present. Here are some examples of block widths for standard pixel formats:

- 8-bit luminance only, block width is 1, the bits per block value is 8.
- 16-bit 1555 RGB, block width is 1, the bits per block value is 16.
- 32-bit 8888 ARGB, block width is 1, the bits per block value is 32.
- 2vuy (CbYCrY), block width is 2, the bits per block value is 32.
- 1-bit bitmap, block width is 8, the bits per block value is 8.
- v210, block width is 6, the bits per block value is 128 .

Available in iOS 4.0 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatBlockHeight`
: The height, in pixels, of the smallest byte-addressable group of pixels (type `CFNumber`). Assumed to be one if this key is not present.

  Available in iOS 4.0 and later.

  Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatBitsPerBlock`
: The number of bits per block. For simple pixel formats, this value is the same as the traditional bits-per-pixel value. This key is mandatory in pixel format descriptions. See the description for `kCVPixelFormatBlockWidth` for examples of bits-per-block values.

  Available in iOS 4.0 and later.

  Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatBlockHorizontalAlignment`
: The horizontal alignment requirements of this format (type `CFNumber`). For example,the alignment for v210 would be '8' here for the horizontal case to match the standard v210 row alignment value of 48. Assumed to be 1 if this key is not present.

  Available in iOS 4.0 and later.

  Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatBlockVerticalAlignment`
: The vertical alignment requirements of this format (type `CFNumber`). Assumed to be 1 if this key is not present.

  Available in iOS 4.0 and later.

  Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatBlackBlock`
: The bit pattern for a block of black pixels (type `CFData`. If absent, black is assumed to be all zeros. If present, this should be `bitsPerPixel` bits long; if `bitsPerPixel` is less than a byte, repeat the bit pattern for the full byte.

  Available in iOS 4.0 and later.

  Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatHorizontalSubsampling`
: Horizontal subsampling information for this plane (type `CFNumber`). Assumed to be 1 if this key is not present.

  Available in iOS 4.0 and later.

  Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatVerticalSubsampling`
: Vertical subsampling information for this plane (type `CFNumber`). Assumed to be 1 if this key is not present.

  Available in iOS 4.0 and later.

  Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatOpenGLFormat`
: The OpenGL format used to describe this image plane (if applicable). See the OpenGL specification for possible values.

  Available in iOS 4.0 and later.

  Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatOpenGLType`
The OpenGL type to describe this image plane (if applicable). See the OpenGL specification for possible values.

Available in iOS 4.0 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatOpenGLInternalFormat`
The OpenGL internal format for this pixel format (if applicable). See the OpenGL specification for possible values.

Available in iOS 4.0 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatCGBitmapInfo`
The Core Graphics bitmap information for this pixel format (if applicable).

Available in iOS 4.0 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatQDCompatibility`
Indicates whether this format is compatible with QuickDraw (type `CFBoolean`).

Available in iOS 4.0 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatCGBitmapContextCompatibility`
Indicates whether this format is compatible with Core Graphics bitmap contexts(type `CFBoolean`).

Available in iOS 4.0 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatCGImageCompatibility`
Indicates whether this format is compatible with the `CGImage` type (type `CFBoolean`).

Available in iOS 4.0 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatOpenGLCompatibility`
Indicates whether this format is compatible with OpenGL (type `CFBoolean`).

Available in iOS 4.0 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatFillExtendedPixelsCallback`
Specifies a custom extended pixel fill algorithm (type `CFData`). See `CVFillExtendedPixelsCallBack` (page 173) and `CVFillExtendedPixelsCallbackData` (page 176) for more information.

Available in iOS 4.0 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatDirect3DCompatibility`
Description forthcoming.

`kCVPixelFormatDirect3DFormat`
Description forthcoming.

`kCVPixelFormatDirect3DInternalFormat`
Description forthcoming.

`kCVPixelFormatDirect3DType`
Description forthcoming.

Discussion
If you need to define a custom pixel format, you must specify these keys in a Core Foundation dictionary. For information about registering your pixel format, see Technical Q&A 1401: Registering Custom Pixel Formats with QuickTime and Core Video.

In most cases you do not need to specify your own pixel format.

SMPTE State Flags

Flags that describe the SMPTE time state.

```
enum{
    kCVSMPTETimeValid       = (1L << 0),
    kCVSMPTETimeRunning     = (1L << 1)
};
```

Constants

`kCVSMPTETimeValid`

The full time is valid.

Available in iOS 4.0 and later.

Declared in `CVBase.h`.

`kCVSMPTETimeRunning`

Time is running.

Available in iOS 4.0 and later.

Declared in `CVBase.h`.

Discussion
You use these values in the `CVSMPTETime` (page 178) structure.

SMPTE Time Types

Constants that describe the type of SMPTE time.

```
enum{
    kCVSMPTETimeType24          = 0,
    kCVSMPTETimeType25          = 1,
    kCVSMPTETimeType30Drop      = 2,
    kCVSMPTETimeType30          = 3,
    kCVSMPTETimeType2997        = 4,
    kCVSMPTETimeType2997Drop    = 5,
    kCVSMPTETimeType60          = 6,
    kCVSMPTETimeType5994        = 7
};
```

Constants

`kCVSMPTETimeType24`

24 frames per second (standard film).

Available in iOS 4.0 and later.

Declared in `CVBase.h`.

`kCVSMPTETimeType25`
: 25 frames per second (standard PAL).

  Available in iOS 4.0 and later.

  Declared in `CVBase.h`.

`kCVSMPTETimeType30Drop`
: 30 drop frame.

  Available in iOS 4.0 and later.

  Declared in `CVBase.h`.

`kCVSMPTETimeType30`
: 30 frames per second.

  Available in iOS 4.0 and later.

  Declared in `CVBase.h`.

`kCVSMPTETimeType2997`
: 29.97 frames per second (standard NTSC).

  Available in iOS 4.0 and later.

  Declared in `CVBase.h`.

`kCVSMPTETimeType2997Drop`
: 29.97 drop frame.

  Available in iOS 4.0 and later.

  Declared in `CVBase.h`.

`kCVSMPTETimeType60`
: 60 frames per second.

  Available in iOS 4.0 and later.

  Declared in `CVBase.h`.

`kCVSMPTETimeType5994`
: 59.94 frames per second.

  Available in iOS 4.0 and later.

  Declared in `CVBase.h`.

Discussion
You use these values in the `CVSMPTETime` (page 178) structure.

Pixel Buffer Locking Flags

Specify the flags to pass to `CVPixelBufferLockBaseAddress` (page 167) and `CVPixelBufferUnlockBaseAddress` (page 171).

```
enum CVPixelBufferLockFlags {
    kCVPixelBufferLock_ReadOnly = 0x00000001,
};
```

Constants

`kCVPixelBufferLock_ReadOnly`

Read-only buffer.

If you are not going to modify the data while you hold the lock, you should set this flag to avoid potentially invalidating any existing caches of the buffer contents. This flag should be passed both to the lock and unlock functions. Non-symmetrical usage of this flag will result in undefined behavior.

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

Discussion

You should set these attributes when adding attachments to a `CVBuffer` object.

Pixel Buffer Pool Attribute Keys

Specify attributes associated with a pixel buffer pool.

```
const CFStringRef kCVImageBufferChromaLocationTopFieldKey;
const CFStringRef kCVImageBufferChromaLocationBottomFieldKey;
const CFStringRef kCVImageBufferChromaLocation_Left;
const CFStringRef kCVImageBufferChromaLocation_Center;
const CFStringRef kCVImageBufferChromaLocation_TopLeft;
const CFStringRef kCVImageBufferChromaLocation_Top;
const CFStringRef kCVImageBufferChromaLocation_BottomLeft;
const CFStringRef kCVImageBufferChromaLocation_Bottom;
const CFStringRef kCVImageBufferChromaLocation_DV420;
```

Constants

`kCVImageBufferChromaLocationTopFieldKey`

Key with one of the following values.

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferChromaLocationBottomFieldKey`

Key with one of the following values.

For progressive images, only `kCVImageBufferChromaLocationTopFieldKey` is used.

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferChromaLocation_Left`

Chroma sample is horizontally co-sited with the left column of luma samples, but centered vertically.

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferChromaLocation_Center`

Chroma sample is fully centered.

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferChromaLocation_TopLeft`
: Chroma sample is co-sited with the top-left luma sample.

  Available in iOS 4.0 and later.

  Declared in `CVImageBuffer.h`.

`kCVImageBufferChromaLocation_Top`
: Chroma sample is horizontally centered, but co-sited with the top row of luma samples.

  Available in iOS 4.0 and later.

  Declared in `CVImageBuffer.h`.

`kCVImageBufferChromaLocation_BottomLeft`
: Chroma sample is co-sited with the bottom-left luma sample.

  Available in iOS 4.0 and later.

  Declared in `CVImageBuffer.h`.

`kCVImageBufferChromaLocation_Bottom`
: Chroma sample is horizontally centered, but co-sited with the bottom row of luma samples.

  Available in iOS 4.0 and later.

  Declared in `CVImageBuffer.h`.

`kCVImageBufferChromaLocation_DV420`
: Cr and Cb samples are alternately co-sited with the left luma samples of the same field.

  Available in iOS 4.0 and later.

  Declared in `CVImageBuffer.h`.

Image Buffer Chroma Subsampling Keys

Describe the format of the original subsampled data before conversion to 422/2vuy. In order to use these tags, the data must have been converted to 4:2:2 via simple pixel replication.

```
const CFStringRef kCVImageBufferChromaSubsamplingKey;
const CFStringRef kCVImageBufferChromaSubsampling_420;
const CFStringRef kCVImageBufferChromaSubsampling_422;
const CFStringRef kCVImageBufferChromaSubsampling_411;
```

Constants

`kCVImageBufferChromaSubsamplingKey`
: Description forthcoming.

  Available in iOS 4.0 and later.

  Declared in `CVImageBuffer.h`.

`kCVImageBufferChromaSubsampling_420`
: Description forthcoming.

  Available in iOS 4.0 and later.

  Declared in `CVImageBuffer.h`.

`kCVImageBufferChromaSubsampling_422`
: Description forthcoming.

  Available in iOS 4.0 and later.

  Declared in `CVImageBuffer.h`.

`kCVImageBufferChromaSubsampling_411`
Description forthcoming.

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

Image Buffer Color Primaries Keys

Describe the color primaries.

```
const CFStringRef kCVImageBufferColorPrimariesKey;
const CFStringRef kCVImageBufferColorPrimaries_ITU_R_709_2;
const CFStringRef kCVImageBufferColorPrimaries_EBU_3213;
const CFStringRef kCVImageBufferColorPrimaries_SMPTE_C;
```

Constants

`kCVImageBufferColorPrimariesKey`
Description forthcoming.

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferColorPrimaries_ITU_R_709_2`
Description forthcoming.

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferColorPrimaries_EBU_3213`
Description forthcoming.

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferColorPrimaries_SMPTE_C`
Description forthcoming.

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

ICC Profile Representation

Represents the ICC profile.

```
const CFStringRef kCVImageBufferICCProfileKey;
```

Constants

`Constant`
Represents the ICC profile.

Image Buffer Transfer Functions

Represents the ICC profile.

```
const CFStringRef kCVImageBufferTransferFunctionKey;
const CFStringRef kCVImageBufferTransferFunction_ITU_R_709_2;
const CFStringRef kCVImageBufferTransferFunction_SMPTE_240M_1995;
const CFStringRef kCVImageBufferTransferFunction_UseGamma;
```

Constants

`kCVImageBufferTransferFunctionKey`

Key with one of the following values, describing the transfer function.

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferTransferFunction_ITU_R_709_2`

Description forthcoming.

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferTransferFunction_SMPTE_240M_1995`

Description forthcoming.

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferTransferFunction_UseGamma`

Description forthcoming.

Available in iOS 4.0 and later.

Declared in `CVImageBuffer.h`.

Pixel Format Types

CoreVideo does not provide support for all of these formats; this list just defines their names.

```
enum {
  kCVPixelFormatType_1Monochrome    = 0x00000001,
  kCVPixelFormatType_2Indexed       = 0x00000002,
  kCVPixelFormatType_4Indexed       = 0x00000004,
  kCVPixelFormatType_8Indexed       = 0x00000008,
  kCVPixelFormatType_1IndexedGray_WhiteIsZero = 0x00000021,
  kCVPixelFormatType_2IndexedGray_WhiteIsZero = 0x00000022,
  kCVPixelFormatType_4IndexedGray_WhiteIsZero = 0x00000024,
  kCVPixelFormatType_8IndexedGray_WhiteIsZero = 0x00000028,
  kCVPixelFormatType_16BE555        = 0x00000010,
  kCVPixelFormatType_16LE555        = 'L555',
  kCVPixelFormatType_16LE5551       = '5551',
  kCVPixelFormatType_16BE565        = 'B565',
  kCVPixelFormatType_16LE565        = 'L565',
  kCVPixelFormatType_24RGB          = 0x00000018,
  kCVPixelFormatType_24BGR          = '24BG',
  kCVPixelFormatType_32ARGB         = 0x00000020,
  kCVPixelFormatType_32BGRA         = 'BGRA',
  kCVPixelFormatType_32ABGR         = 'ABGR',
  kCVPixelFormatType_32RGBA         = 'RGBA',
  kCVPixelFormatType_64ARGB         = 'b64a',
  kCVPixelFormatType_48RGB          = 'b48r',
  kCVPixelFormatType_32AlphaGray    = 'b32a',
  kCVPixelFormatType_16Gray         = 'b16g',
  kCVPixelFormatType_422YpCbCr8     = '2vuy',
  kCVPixelFormatType_4444YpCbCrA8   = 'v408',
  kCVPixelFormatType_4444YpCbCrA8R  = 'r408',
  kCVPixelFormatType_444YpCbCr8     = 'v308',
  kCVPixelFormatType_422YpCbCr16    = 'v216',
  kCVPixelFormatType_422YpCbCr10    = 'v210',
  kCVPixelFormatType_444YpCbCr10    = 'v410',
  kCVPixelFormatType_420YpCbCr8Planar = 'y420',
  kCVPixelFormatType_420YpCbCr8PlanarFullRange    = 'f420',
  kCVPixelFormatType_422YpCbCr_4A_8BiPlanar = 'a2vy',
  kCVPixelFormatType_420YpCbCr8BiPlanarVideoRange = '420v',
  kCVPixelFormatType_420YpCbCr8BiPlanarFullRange  = '420f',
  kCVPixelFormatType_422YpCbCr8_yuvs = 'yuvs',
  kCVPixelFormatType_422YpCbCr8FullRange = 'yuvf',
};
```

Constants

`kCVPixelFormatType_1Monochrome`

1 bit indexed.

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_2Indexed`

2 bit indexed.

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_4Indexed`

4 bit indexed.

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_8Indexed`
: 8 bit indexed.

  Available in iOS 4.0 and later.

  Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_1IndexedGray_WhiteIsZero`
: 1 bit indexed gray, white is zero.

  Available in iOS 4.0 and later.

  Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_2IndexedGray_WhiteIsZero`
: 2 bit indexed gray, white is zero.

  Available in iOS 4.0 and later.

  Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_4IndexedGray_WhiteIsZero`
: 4 bit indexed gray, white is zero.

  Available in iOS 4.0 and later.

  Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_8IndexedGray_WhiteIsZero`
: 8 bit indexed gray, white is zero.

  Available in iOS 4.0 and later.

  Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_16BE555`
: 16 bit BE RGB 555.

  Available in iOS 4.0 and later.

  Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_16LE555`
: 16 bit LE RGB 555.

  Available in iOS 4.0 and later.

  Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_16LE5551`
: 16 bit LE RGB 5551.

  Available in iOS 4.0 and later.

  Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_16BE565`
: 16 bit BE RGB 565.

  Available in iOS 4.0 and later.

  Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_16LE565`
: 16 bit LE RGB 565.

  Available in iOS 4.0 and later.

  Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_24RGB`
: 24 bit RGB.

  Available in iOS 4.0 and later.

  Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_24BGR`
: 24 bit BGR.

  Available in iOS 4.0 and later.

  Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_32ARGB`
: 32 bit ARGB.

  Available in iOS 4.0 and later.

  Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_32BGRA`
: 32 bit BGRA.

  Available in iOS 4.0 and later.

  Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_32ABGR`
: 32 bit ABGR.

  Available in iOS 4.0 and later.

  Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_32RGBA`
: 32 bit RGBA.

  Available in iOS 4.0 and later.

  Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_64ARGB`
: 64 bit ARGB, 16-bit big-endian samples.

  Available in iOS 4.0 and later.

  Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_48RGB`
: 48 bit RGB, 16-bit big-endian samples.

  Available in iOS 4.0 and later.

  Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_32AlphaGray`
: 32 bit AlphaGray, 16-bit big-endian samples, black is zero.

  Available in iOS 4.0 and later.

  Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_16Gray`
: 16 bit Grayscale, 16-bit big-endian samples, black is zero.

  Available in iOS 4.0 and later.

  Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_422YpCbCr8`
Component Y'CbCr 8-bit 4:2:2, ordered Cb Y'0 Cr Y'1.

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_4444YpCbCrA8`
Component Y'CbCrA 8-bit 4:4:4:4, ordered Cb Y' Cr A.

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_4444YpCbCrA8R`
Component Y'CbCrA 8-bit 4:4:4:4, rendering format. Full range alpha, zero biased YUV, ordered A Y' Cb Cr.

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_444YpCbCr8`
Component Y'CbCr 8-bit 4:4:4.

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_422YpCbCr16`
Component Y'CbCr 10,12,14,16-bit 4:2:2.

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_422YpCbCr10`
Component Y'CbCr 10-bit 4:2:2.

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_444YpCbCr10`
Component Y'CbCr 10-bit 4:4:4.

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_420YpCbCr8Planar`
Planar Component Y'CbCr 8-bit 4:2:0. `baseAddr` points to a big-endian `CVPlanarPixelBufferInfo_YCbCrPlanar` struct.

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_420YpCbCr8PlanarFullRange`
Planar Component Y'CbCr 8-bit 4:2:0, full range. `baseAddr` points to a big-endian `CVPlanarPixelBufferInfo_YCbCrPlanar` struct.

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_422YpCbCr_4A_8BiPlanar`
First plane: Video-range Component Y'CbCr 8-bit 4:2:2, ordered Cb Y'0 Cr Y'1; second plane: alpha 8-bit 0-255.

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_420YpCbCr8BiPlanarVideoRange`
Bi-Planar Component Y'CbCr 8-bit 4:2:0, video-range (luma=[16,235] chroma=[16,240]). `baseAddr` points to a big-endian `CVPlanarPixelBufferInfo_YCbCrBiPlanar` struct.

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_420YpCbCr8BiPlanarFullRange`
Bi-Planar Component Y'CbCr 8-bit 4:2:0, full-range (luma=[0,255] chroma=[1,255]). `baseAddr` points to a big-endian `CVPlanarPixelBufferInfo_YCbCrBiPlanar` struct.

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_422YpCbCr8_yuvs`
Component Y'CbCr 8-bit 4:2:2, ordered Y'0 Cb Y'1 Cr.

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelFormatType_422YpCbCr8FullRange`
Component Y'CbCr 8-bit 4:2:2, full range, ordered Y'0 Cb Y'1 Cr.

Available in iOS 4.0 and later.

Declared in `CVPixelBuffer.h`.

Result Codes

The table below lists the result codes returned for Core Video. Note that these result codes are of type `CVReturn`, not type `OSErr`.

Result Code	Value	Description
`kCVReturnSuccess`	0	No error Available in iOS 4.0 and later.
`kCVReturnFirst`	-6660	Placeholder to mark the beginning of Core Video result codes (not returned by any functions). Available in iOS 4.0 and later.
`kCVReturnError`	-6660	An otherwise undefined error occurred. Available in iOS 4.0 and later.

Result Code	Value	Description
`kCVReturnInvalidArgument`	-6661	Invalid function parameter. For example, out of range or the wrong type. Available in iOS 4.0 and later.
`kCVReturnAllocationFailed`	-6662	Memory allocation for a buffer or buffer pool failed. Available in iOS 4.0 and later.
`kCVReturnInvalidDisplay`	-6670	The display specified when creating a display link is invalid. Available in iOS 4.0 and later.
`kCVReturnDisplayLinkAlreadyRunning`	-6671	The specified display link is already running. Available in iOS 4.0 and later.
`kCVReturnDisplayLinkNotRunning`	-6672	The specified display link is not running. Available in iOS 4.0 and later.
`kCVReturnDisplayLinkCallbacksNotSet`	-6673	No callback registered for the specified display link. You must set either the output callback or both the render and display callbacks. Available in iOS 4.0 and later.
`kCVReturnInvalidPixelFormat`	-6680	The buffer does not support the specified pixel format. Available in iOS 4.0 and later.
`kCVReturnInvalidSize`	-6681	The buffer cannot support the requested buffer size (usually too big). Available in iOS 4.0 and later.
`kCVReturnInvalidPixelBufferAttributes`	-6682	A buffer cannot be created with the specified attributes. Available in iOS 4.0 and later.
`kCVReturnPixelBufferNotOpenGLCompatible`	-6683	The pixel buffer is not compatible with OpenGL due to an unsupported buffer size, pixel format, or attribute. Available in iOS 4.0 and later.
`kCVReturnPoolAllocationFailed`	-6690	Allocation for a buffer pool failed, most likely due to a lack of resources. Check to make sure your parameters are in range. Available in iOS 4.0 and later.

Result Code	Value	Description
`kCVReturnInvalidPoolAttributes`	-6691	A buffer pool cannot be created with the specified attributes. Available in iOS 4.0 and later.
`kCVReturnLast`	-6699	Placeholder to mark the end of Core Video result codes (not returned by any functions). Available in iOS 4.0 and later.

Core Animation Function Reference

Framework:	QuartzCore/QuartzCore.h
Declared in	CABase.h CATransform3D.h

Overview

Functions by Task

Timing Functions

`CACurrentMediaTime` (page 210)
Returns the current absolute time, in seconds.

Transform Functions

`CATransform3DIsIdentity` (page 211)
Returns a Boolean value that indicates whether the transform is the identity transform.

`CATransform3DEqualToTransform` (page 210)
Returns a Boolean value that indicates whether the two transforms are exactly equal.

`CATransform3DMakeTranslation` (page 213)
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 212)
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 212)
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 213)
Translate 't' by '(tx, ty, tz)' and return the result: t' = translate(tx, ty, tz) * t.

`CATransform3DScale` (page 213)
Scale 't' by '(sx, sy, sz)' and return the result: t' = scale(sx, sy, sz) * t.

`CATransform3DRotate` (page 213)
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 210)
: Concatenate 'b' to 'a' and return the result: t' = a * b.

`CATransform3DInvert` (page 211)
: Invert 't' and return the result. Returns the original matrix if 't' has no inverse.

`CATransform3DMakeAffineTransform` (page 212)
: Return a transform with the same effect as affine transform 'm'.

`CATransform3DIsAffine` (page 211)
: Returns true if 't' can be exactly represented by an affine transform.

`CATransform3DGetAffineTransform` (page 211)
: 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.

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.

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.

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
2009-09-09	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.