

---

# Core Animation Reference Collection

Graphics & Animation: Animation



2009-07-15



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

.Mac is a registered service mark of Apple Inc.

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

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**      **Core Animation Reference Collection** 15

---

**Part I**              **Classes** 17

---

**Chapter 1**         **CAAnimation Class Reference** 19

---

Overview 19  
Tasks 19  
Properties 20  
Class Methods 22  
Instance Methods 23  
Delegate Methods 23

**Chapter 2**         **CAAnimationGroup Class Reference** 25

---

Overview 25  
Tasks 26  
Properties 26

**Chapter 3**         **CABasicAnimation Class Reference** 27

---

Overview 27  
Tasks 28  
Properties 28

**Chapter 4**         **CAConstraint Class Reference** 31

---

Overview 31  
Tasks 32  
Properties 32  
Class Methods 34  
Instance Methods 36  
Constants 36

**Chapter 5**         **CAConstraintLayoutManager Class Reference** 39

---

Overview 39  
Tasks 40  
Class Methods 40

**Chapter 6**      **CAEmitterCell Class Reference**    **41**

---

Overview 41  
Tasks 41  
Properties 44  
Class Methods 57  
Instance Methods 58

**Chapter 7**      **CAEmitterLayer Class Reference**    **61**

---

Overview 61  
Tasks 61  
Properties 62  
Constants 68

**Chapter 8**      **CAGradientLayer Class Reference**    **71**

---

Overview 71  
Tasks 71  
Properties 71  
Constants 73

**Chapter 9**      **CAKeyframeAnimation Class Reference**    **75**

---

Overview 75  
Tasks 75  
Properties 76  
Constants 78

**Chapter 10**      **CALayer Class Reference**    **81**

---

Overview 81  
Tasks 81  
Properties 87  
Class Methods 106  
Instance Methods 108  
Delegate Methods 127  
Constants 128

**Chapter 11**      **CAMediaTimingFunction Class Reference**    **135**

---

Overview 135  
Tasks 135  
Class Methods 136  
Instance Methods 137  
Constants 138

**Chapter 12**      **CAOpenGLLayer Class Reference**   141

---

Overview 141  
Tasks 141  
Properties 142  
Instance Methods 143

**Chapter 13**      **CAPropertyAnimation Class Reference**   147

---

Overview 147  
Tasks 147  
Properties 148  
Class Methods 149

**Chapter 14**      **CARenderer Class Reference**   151

---

Overview 151  
Tasks 151  
Properties 152  
Class Methods 153  
Instance Methods 153

**Chapter 15**      **CAReplicatorLayer Class Reference**   157

---

Overview 157  
Tasks 157  
Properties 158

**Chapter 16**      **CAScrollLayer Class Reference**   163

---

Overview 163  
Tasks 163  
Properties 164  
Instance Methods 164  
Constants 165

**Chapter 17**      **CAShapeLayer Class Reference**   167

---

Overview 167  
Tasks 167  
Properties 168  
Constants 172

**Chapter 18**      **CATextLayer Class Reference**   175

---

Overview 175

Tasks 176  
Properties 176  
Constants 179

---

**Chapter 19      CATransaction Class Reference 183**

---

Overview 183  
Tasks 184  
Class Methods 185  
Constants 192

---

**Chapter 20      CATransformLayer Class Reference 193**

---

Overview 193  
Tasks 194  
Instance Methods 194

---

**Chapter 21      CATransition Class Reference 195**

---

Overview 195  
Tasks 195  
Properties 196  
Constants 198

---

**Chapter 22      CAValueFunction Class Reference 201**

---

Overview 201  
Tasks 201  
Properties 202  
Class Methods 202  
Constants 203

---

**Chapter 23      CIFilter Core Animation Additions 207**

---

Overview 207  
Tasks 207  
Properties 208  
Instance Methods 208

---

**Chapter 24      NSValue Core Animation Additions 209**

---

Overview 209  
Tasks 209  
Class Methods 209  
Instance Methods 210

---

**Chapter 25**      **QCCompositionLayer Class Reference** 211

---

Overview 211  
Tasks 212  
Class Methods 212  
Instance Methods 213

---

**Part II**            **Protocols** 215

---

---

**Chapter 26**      **CAAction Protocol Reference** 217

---

Overview 217  
Tasks 217  
Instance Methods 217

---

**Chapter 27**      **CALayoutManager Protocol Reference** 219

---

Overview 219  
Tasks 219  
Instance Methods 219

---

**Chapter 28**      **CAMediaTiming Protocol Reference** 223

---

Overview 223  
Tasks 223  
Properties 224  
Constants 227

---

**Part III**            **Other References** 229

---

---

**Chapter 29**      **Core Animation Function Reference** 231

---

Overview 231  
Functions by Task 231  
Functions 232

---

**Drawing** 237

---

Drawing Layer Content With Application Kit Classes 237

---

**What Is Core Animation?** 239

---

Core Animation Classes 239

---

## Timing 245

---

Using a Single Timing Function For a Keyframe Animation 245

---

## Core Animation Rendering Architecture 247

---

---

## Layer Geometry and Transforms 249

---

Layer Coordinate System 249  
Specifying a Layer's Geometry 249  
Transforming a Layer's Geometry 252

---

## Layer-Tree Hierarchy 257

---

What Is a Layer-Tree Hierarchy? 257  
Displaying Layers in Views 257  
Adding and Removing Layers from a Hierarchy 258  
Repositioning and Resizing Layers 258  
Clipping Sublayers 260

---

## Providing Layer Content 261

---

Providing CALayer Content 261  
Positioning Content Within a Layer 264

---

## Animation 267

---

Animation Classes and Timing 267  
Implicit Animation 267  
Explicit Animation 268  
Starting and Stopping Explicit Animations 269

---

## Layer Actions 271

---

The Role of Action Objects 271  
Defined Search Pattern for Action Keys 271  
Adopting the CAAction Protocol 272  
Overriding an Implied Animation 272  
Temporarily Disabling Actions 273

---

## Transactions 275

---

Implicit transactions 275  
Explicit Transactions 275



---

## Laying Out Core Animation Layers 279

---

Constraints Layout Manager 279

---

## Core Animation Extensions To Key-Value Coding 283

---

Key-Value Coding Compliant Container Classes 283

Default Value Support 283

Wrapping Conventions 284

Key Path Support for Structure Fields 284

---

## Layer Style Properties 287

---

Geometry Properties 287

Background Properties 288

Layer Content 288

Sublayers Content 289

Border Attributes 290

Filters Property 291

Shadow Properties 291

Opacity Property 292

Composite Property 293

Mask Properties 293

---

## Example: Core Animation Kiosk Menu Style Application 295

---

The User Interface 295

Examining the Application's Nib 297

Examining the Application's Code 298

Performance Considerations 305

---

## Animatable Properties 307

---

CALayer Animatable Properties 307

CIFilter Animatable Properties 309

---

## Document Revision History 311

---



# Figures, Tables, and Listings

## Chapter 17

### **CAShapeLayer Class Reference 167**

---

Figure 17-1 Line cap styles 169

Figure 17-2 Line join styles 171

### **Drawing 237**

---

Listing 1 Drawing into a layer using Application Kit classes 237

### **What Is Core Animation? 239**

---

Figure 1 Core Animation class hierarchy 240

### **Timing 245**

---

Listing 1 Using a single timing function for a keyframe animation 245

### **Core Animation Rendering Architecture 247**

---

Figure 1 Core Animation Rendering Architecture 247

### **Layer Geometry and Transforms 249**

---

Figure 1 CALayer geometry properties 250

Figure 2 Three anchorPoint values 251

Figure 3 Layer Origin of (0.5,0.5) 251

Figure 4 Layer Origin of (0.0,0.0) 252

Table 1 CATransform3D transform functions for translation, rotation, and scaling 253

Table 2 CATransform3D transform functions for CGAffineTransform conversion 254

Table 3 CATransform3D transform functions for testing equality 254

Table 4 CATransform3D key paths 255

Listing 1 CATransform3D structure 254

Listing 2 Modifying the CATransform3D data structure directly 254

### **Layer-Tree Hierarchy 257**

---

Figure 1 Layer autoresizing mask constants 260

Figure 2 Example Values of the masksToBounds property 260

Table 1 Layer-tree management methods. 258

Table 2 Autoresizing mask values and descriptions 259

Listing 1 Inserting a layer into a view 257

---

## Providing Layer Content 261

---

Figure 1	Position constants for a layer's contentsGravity property	265
Figure 2	Scaling constants for a layer's contentsGravity property	265
Table 1	Positioning constants for a layer's contentsGravity property	264
Table 2	Scaling Constants For A Layer's contentsGravity Property	265
Listing 1	Setting a layer's contents property	261
Listing 2	Example implementation of the delegate method displayLayer:	262
Listing 3	Example implementation of the delegate method drawLayer:inContext:	262
Listing 4	Example override of the CALayer display method	263
Listing 5	Example override of the CALayer drawInContext: method	263

---

## Animation 267

---

Listing 1	Implicitly animating a layer's position property	268
Listing 2	Implicitly animating multiple properties of multiple layers	268
Listing 3	Explicit animation	268
Listing 4	Continuous explicit animation example	269

---

## Layer Actions 271

---

Table 1	Action triggers and their corresponding identifiers	271
Listing 1	runActionForKey:object:arguments: implementation that initiates an animation	272
Listing 2	Implied animation for the contents property	272
Listing 3	Implied animation for the sublayers property	273

---

## Transactions 275

---

Listing 1	Animation using an implicit transaction	275
Listing 2	Temporarily disabling a layer's actions	276
Listing 3	Overriding the animation duration	276
Listing 4	Nesting explicit transactions	276

---

## Laying Out Core Animation Layers 279

---

Figure 1	Constraint layout manager attributes	279
Figure 2	Example constraints based layout	280
Listing 1	Configuring a layer's constraints	280

---

## Core Animation Extensions To Key-Value Coding 283

---

Listing 1	Example implementation of defaultValueForKey:	283
-----------	---	-----

## Layer Style Properties 287

---

Figure 1	Layer geometry 287
Figure 2	Layer with background color 288
Figure 3	Layer displaying a content image 289
Figure 4	Layer displaying the sublayers content 289
Figure 5	Layer displaying the border attributes content 290
Figure 6	Layer displaying the filters properties 291
Figure 7	Layer displaying the shadow properties 292
Figure 8	Layer including the opacity property 292
Figure 9	Layer composited using the compositingFilter property 293
Figure 10	Layer composited with the mask property 294

## Example: Core Animation Kiosk Menu Style Application 295

---

Figure 1	Core Animation Kiosk Menu Interface 296
Figure 2	Layer Hierarchy For QCCoreAnimationKioskStyleMenu Application 297
Figure 3	MainMenu.nib File 298
Figure 4	Alternate Interface With Black Background. 306
Listing 1	SelectionView.h File Listing 299
Listing 2	Implementation of awakeFromNib 300
Listing 3	Configuration of the Background rootLayer 301
Listing 4	Setup menuLayers Array. The Selectable Menu Items. 302
Listing 5	Setup selectionLayer. Used to Display the Currently Selected Item. 303
Listing 6	Handle Changes in Selection 304
Listing 7	Handling Up and Down Key Presses 304
Listing 8	Dealloc and Cleanup 305

## Animatable Properties 307

---

Table 1	Default Implied Basic Animation 309
Table 2	Default Implied Transition 309



# Core Animation Reference Collection

---

<b>Framework</b>	/System/Library/Frameworks/QuartzCore.frameworks
<b>Header file directories</b>	/System/Library/Frameworks/QuartzCore.frameworks/Headers
<b>Companion guide</b>	Core Animation Programming Guide

<b>Declared in</b>	CAAnimation.h
	CABase.h
	CACIFilterAdditions.h
	CAConstraintLayoutManager.h
	CAEmitterCell.h
	CAEmitterLayer.h
	CAGradientLayer.h
	CALayer.h
	CAMediaTiming.h
	CAMediaTimingFunction.h
	CAOpenGLLayer.h
	CARenderer.h
	CAReplicatorLayer.h
	CAScrollLayer.h
	CAShapeLayer.h
	CATextLayer.h
	CATransaction.h
	CATransform3D.h
	CAValueFunction.h
	QCCompositionLayer.h

This collection of documents provides the API reference for Core Animation. Core Animation provides animation and display hierarchy capabilities to applications. For more details, see *Core Animation Programming Guide*.

## INTRODUCTION

Core Animation Reference Collection



# Classes

---



# CAAnimation Class Reference

---

<b>Inherits from</b>	NSObject
<b>Conforms to</b>	NSCoding NSCopying CAAction CAMediaTiming NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.5 and later.
<b>Declared in</b>	CAAnimation.h
<b>Companion guides</b>	Core Animation Programming Guide Core Animation Cookbook
<b>Related sample code</b>	AnimatedTableView

## Overview

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

## Tasks

### Archiving Properties

- `shouldArchiveValueForKey:` (page 23)  
Specifies whether the value of the property for a given key is archived.

### Providing Default Values for Properties

- + `defaultValueForKey:` (page 22)  
Specifies the default value of the property with the specified key.

## Creating an Animation

+ `animation` (page 22)

Creates and returns a new `CAAnimation` instance.

## Animation Attributes

`removedOnCompletion` (page 21) *property*

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

- `isRemovedOnCompletion` (page 23)

A synthesized accessor for the `removedOnCompletion` (page 21) property.

`timingFunction` (page 21) *property*

An optional timing function defining the pacing of the animation.

## Getting and Setting the Delegate

`delegate` (page 20) *property*

Specifies the receiver's delegate object.

## Animation Progress

- `animationDidStart:` (page 23) *delegate method*

Called when the animation begins its active duration.

- `animationDidStop:finished:` (page 24) *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 Mac OS X v10.5 and later.

**Related Sample Code**

`AnimatedTableView`

**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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Related Sample Code**

`CoreAnimationKioskStyleMenu`

**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 Mac OS X v10.5 and later.

#### Related Sample Code

`AnimatedTableView`

`CocoaSlides`

`CoreAnimationKioskStyleMenu`

`TrackBall`

#### 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 Mac OS X v10.5 and later.

#### Declared In

`CAAnimation.h`

## Instance Methods

### **isRemovedOnCompletion**

A synthesized accessor for the `removedOnCompletion` (page 21) property.

- (BOOL)isRemovedOnCompletion

#### **See Also**

[@property removedOnCompletion](#) (page 21)

### **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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Declared In**

CAAnimation.h



# CAAnimationGroup Class Reference

---

<b>Inherits from</b>	CAAnimation : NSObject
<b>Conforms to</b>	NSCoding (CAAnimation) NSCopying (CAAnimation) CAAction (CAAnimation) CAMediaTiming (CAAnimation) NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.5 and later.
<b>Declared in</b>	CAAnimation.h
<b>Companion guides</b>	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 26) array are currently ignored. The `CAAnimationGroup` delegate does receive these messages.

**Note:** The `delegate` and `removedOnCompletion` properties of animations in the `animations` (page 26) property are currently ignored.

## Tasks

### Grouped Animations

`animations` (page 26) *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 Mac OS X v10.5 and later.

#### **Declared In**

`CAAnimation.h`

# CABasicAnimation Class Reference

---

<b>Inherits from</b>	CAPropertyAnimation : CAAnimation : NSObject
<b>Conforms to</b>	NSCoding (CAAnimation) NSCopying (CAAnimation) CAAction (CAAnimation) CAMediaTiming (CAAnimation) NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.5 and later.
<b>Declared in</b>	CAAnimation.h
<b>Companion guides</b>	Core Animation Programming Guide Core Animation Cookbook
<b>Related sample code</b>	AnimatedTableView CocoaSlides CoreAnimationKioskStyleMenu TrackBall

## Overview

`CABasicAnimation` provides basic, single-keyframe animation capabilities for a layer property. You create an instance of `CABasicAnimation` using the inherited `animationWithKeyPath:` (page 149) method, specifying the key path of the property to be animated in the render tree.

## Setting Interpolation Values

---

The `fromValue` (page 29), `byValue` (page 28) and `toValue` (page 29) 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 29) and `toValue` (page 29) are non-`nil`. Interpolates between `fromValue` (page 29) and `toValue` (page 29).
- `fromValue` (page 29) and `byValue` (page 28) are non-`nil`. Interpolates between `fromValue` (page 29) and (`fromValue` (page 29) + `byValue` (page 28)).

- `byValue` (page 28) and `toValue` (page 29) are non-`nil`. Interpolates between (`toValue` (page 29) - `byValue` (page 28)) and `toValue` (page 29).
- `fromValue` (page 29) is non-`nil`. Interpolates between `fromValue` (page 29) and the current presentation value of the property.
- `toValue` (page 29) is non-`nil`. Interpolates between the current value of `keyPath` in the target layer's presentation layer and `toValue` (page 29).
- `byValue` (page 28) is non-`nil`. Interpolates between the current value of `keyPath` in the target layer's presentation layer and that value plus `byValue` (page 28).
- 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 29) *property*

Defines the value the receiver uses to start interpolation.

`toValue` (page 29) *property*

Defines the value the receiver uses to end interpolation.

`byValue` (page 28) *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 27) for details on how `byValue` interacts with the other interpolation values.

#### Availability

Available in Mac OS X v10.5 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 27) for details on how `fromValue` interacts with the other interpolation values.

### Availability

Available in Mac OS X v10.5 and later.

### Related Sample Code

`AnimatedTableView`

`CoreAnimationKioskStyleMenu`

### Declared In

`CAAnimation.h`

## toValue

Defines the value the receiver uses to end interpolation.

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

### Discussion

See [“Setting Interpolation Values”](#) (page 27) for details on how `toValue` interacts with the other interpolation values.

### Availability

Available in Mac OS X v10.5 and later.

### Related Sample Code

`CoreAnimationKioskStyleMenu`

### Declared In

`CAAnimation.h`



# CAConstraint Class Reference

---

<b>Inherits from</b>	NSObject
<b>Conforms to</b>	NSCoding NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.5 and later.
<b>Declared in</b>	CAConstraintLayoutManager.h
<b>Companion guides</b>	Core Animation Programming Guide Core Animation Cookbook
<b>Related sample code</b>	CoreAnimationKioskStyleMenu

## Overview

`CAConstraint` represents a single layout constraint between two layers. Each `CAConstraint` instance encapsulates one geometry relationship between two layers on the same axis.

Sibling layers are referenced by name, using the `name` property of each layer. The special name `superlayer` is used to refer to the layer's superlayer.

For example, to specify that a layer should be horizontally centered in its superview you would use the following:

```
theConstraint=[CAConstraint constraintWithAttribute:kCAConstraintMidX
                                             relativeTo:@"superlayer"
                                             attribute:kCAConstraintMidX];
```

A maximum of two relationships must be specified per axis. If you specify constraints for the left and right edges of a layer, the width will vary. If you specify constraints for the left edge and the width, the right edge of the layer will move relative to the superlayer's frame. Often you'll specify only a single edge constraint, the layer's size in the same axis will be used as the second relationship.

**Important:** It is possible to create constraints that result in circular references to the same attributes. In cases where the layout is unable to be computed the behavior is undefined.

## Tasks

### Create a New Constraint

- + `constraintWithAttribute:relativeTo:attribute:scale:offset:` (page 35)  
Creates and returns an `CAConstraint` object with the specified parameters.
- + `constraintWithAttribute:relativeTo:attribute:offset:` (page 34)  
Creates and returns an `CAConstraint` object with the specified parameters.
- + `constraintWithAttribute:relativeTo:attribute:` (page 34)  
Creates and returns an `CAConstraint` object with the specified parameters.
- `initWithAttribute:relativeTo:attribute:scale:offset:` (page 36)  
Returns an `CAConstraint` object with the specified parameters. Designated initializer.

### Accessing Constraint Values

- `attribute` (page 32) *property*  
The attribute the constraint affects. (read-only)
- `offset` (page 33) *property*  
Offset value of the constraint attribute. (read-only)
- `scale` (page 33) *property*  
Scale factor of the constraint attribute. (read-only)
- `sourceAttribute` (page 33) *property*  
The constraint attribute of the layer the receiver is calculated relative to (read-only)
- `sourceName` (page 33) *property*  
Name of the layer that the constraint is calculated relative to. (read-only)

## Properties

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

### attribute

The attribute the constraint affects. (read-only)

```
@property(readonly) CAConstraintAttribute attribute
```

#### Availability

Available in Mac OS X v10.6 and later.



**Declared In**

CAConstraintLayoutManager.h

**offset**

Offset value of the constraint attribute. (read-only)

@property(readonly) CGFloat offset

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAConstraintLayoutManager.h

**scale**

Scale factor of the constraint attribute. (read-only)

@property(readonly) CGFloat scale

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAConstraintLayoutManager.h

**sourceAttribute**

The constraint attribute of the layer the receiver is calculated relative to (read-only)

@property(readonly) CAConstraintAttribute sourceAttribute

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAConstraintLayoutManager.h

**sourceName**

Name of the layer that the constraint is calculated relative to. (read-only)

@property(readonly) NSString \*sourceName

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAConstraintLayoutManager.h

## Class Methods

### **constraintWithAttribute:relativeTo:attribute:**

Creates and returns an `CAConstraint` object with the specified parameters.

```
+ (id)constraintWithAttribute:(CAConstraintAttribute)attr relativeTo:(NSString
*)srcLayer attribute:(CAConstraintAttribute)srcAttr
```

#### Parameters

*attr*

The attribute of the layer for which to create a new constraint.

*srcLayer*

The name of the layer that this constraint is calculated relative to.

*srcAttr*

The attribute of *srcLayer* the constraint is calculated relative to.

#### Return Value

A new `CAConstraint` object with the specified parameters. The scale of the constraint is set to 1.0. The offset of the constraint is set to 0.0.

#### Discussion

The value for the constraint is calculated is *srcAttr*.

#### Availability

Available in Mac OS X v10.5 and later.

#### Related Sample Code

CoreAnimationKioskStyleMenu

#### Declared In

CAConstraintLayoutManager.h

### **constraintWithAttribute:relativeTo:attribute:offset:**

Creates and returns an `CAConstraint` object with the specified parameters.

```
+ (id)constraintWithAttribute:(CAConstraintAttribute)attr relativeTo:(NSString
*)srcLayer attribute:(CAConstraintAttribute)srcAttr offset:(CGFloat)offset
```

#### Parameters

*attr*

The attribute of the layer for which to create a new constraint.

*srcLayer*

The name of the layer that this constraint is calculated relative to.

*srcAttr*

The attribute of *srcLayer* the constraint is calculated relative to.

*offset*

The offset added to the value of *srcAttr*.

**Return Value**

A new `CAConstraint` object with the specified parameters. The scale of the constraint is set to 1.0.

**Discussion**

The value for the constraint is calculated as  $(srcAttr + offset)$ .

**Availability**

Available in Mac OS X v10.5 and later.

**Related Sample Code**

`CoreAnimationKioskStyleMenu`

**Declared In**

`CAConstraintLayoutManager.h`

**constraintWithAttribute:relativeTo:attribute:scale:offset:**

Creates and returns an `CAConstraint` object with the specified parameters.

```
+ (id)constraintWithAttribute:(CAConstraintAttribute)attr relativeTo:(NSString
    *)srcLayer attribute:(CAConstraintAttribute)srcAttr scale:(CGFloat)scale
    offset:(CGFloat)offset
```

**Parameters**

*attr*

The attribute of the layer for which to create a new constraint.

*srcLayer*

The name of the layer that this constraint is calculated relative to.

*srcAttr*

The attribute of *srcLayer* the constraint is calculated relative to.

*scale*

The amount to scale the value of *srcAttr*.

*offset*

The offset from the *srcAttr*.

**Return Value**

A new `CAConstraint` object with the specified parameters.

**Discussion**

The value for the constraint is calculated as  $((srcAttr * scale) + offset)$ .

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

`CAConstraintLayoutManager.h`

## Instance Methods

### **initWithAttribute:relativeTo:attribute:scale:offset:**

Returns an `CAConstraint` object with the specified parameters. Designated initializer.

```
- (id)initWithAttribute:(CAConstraintAttribute)attr relativeTo:(NSString *)srcLayer
    attribute:(CAConstraintAttribute)srcAttr scale:(CGFloat)scale
    offset:(CGFloat)offset
```

#### **Parameters**

*attr*

The attribute of the layer for which to create a new constraint.

*srcLayer*

The name of the layer that this constraint is calculated relative to.

*srcAttr*

The attribute of *srcLayer* the constraint is calculated relative to.

*scale*

The amount to scale the value of *srcAttr*.

*offset*

The offset added to the value of *srcAttr*.

#### **Return Value**

An initialized constraint object using the specified parameters.

#### **Discussion**

The value for the constraint is calculated as  $(srcAttr * scale) + offset$ .

#### **Availability**

Available in Mac OS X v10.5 and later.

#### **Declared In**

`CAConstraintLayoutManager.h`

## Constants

### **CAConstraintAttribute**

These constants represent the geometric edge or axis of a constraint.

```
enum _CAConstraintAttribute
{
    kCAConstraintMinX,
    kCAConstraintMidX,
    kCAConstraintMaxX,
    kCAConstraintWidth,
    kCAConstraintMinY,
    kCAConstraintMidY,
    kCAConstraintMaxY,
    kCAConstraintHeight,
};
```

### Constants

`kCAConstraintMinX`

The left edge of a layer's frame.

Available in Mac OS X v10.5 and later.

Declared in `CAConstraintLayoutManager.h`.

`kCAConstraintMidX`

The horizontal location of the center of a layer's frame.

Available in Mac OS X v10.5 and later.

Declared in `CAConstraintLayoutManager.h`.

`kCAConstraintMaxX`

The right edge of a layer's frame.

Available in Mac OS X v10.5 and later.

Declared in `CAConstraintLayoutManager.h`.

`kCAConstraintWidth`

The width of a layer.

Available in Mac OS X v10.5 and later.

Declared in `CAConstraintLayoutManager.h`.

`kCAConstraintMinY`

The bottom edge of a layer's frame.

Available in Mac OS X v10.5 and later.

Declared in `CAConstraintLayoutManager.h`.

`kCAConstraintMidY`

The vertical location of the center of a layer's frame.

Available in Mac OS X v10.5 and later.

Declared in `CAConstraintLayoutManager.h`.

`kCAConstraintMaxY`

The top edge of a layer's frame.

Available in Mac OS X v10.5 and later.

Declared in `CAConstraintLayoutManager.h`.

`kCAConstraintHeight`

The height of a layer.

Available in Mac OS X v10.5 and later.

Declared in `CAConstraintLayoutManager.h`.

**Declared In**

CAConstraint.h

**Constraint Attribute Type**

The constraint attribute type.

```
typedef int CAConstraintAttribute;
```

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CAConstraintLayoutManager.h

# CAConstraintLayoutManager Class Reference

---

<b>Inherits from</b>	NSObject
<b>Conforms to</b>	NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.5 and later.
<b>Declared in</b>	CAConstraintLayoutManager.h
<b>Companion guides</b>	Core Animation Programming Guide Core Animation Cookbook
<b>Related sample code</b>	Core Animation QuickTime Layer CoreAnimationKioskStyleMenu

## Overview

`CAConstraintLayoutManager` provides a constraint-based layout manager.

Constraint-based layout allows you to describe the position and size of a layer by specifying relationships between a layer and its sibling layers or its superlayer. The relationships are represented by instances of the `CAConstraint` class that are stored in an array in the layer's `constraints` property. You add constraints for a layer using its `addConstraint:` (page 110) method. Each `CAConstraint` instance encapsulates one geometry relationship between two layers. Layout is then performed by fetching the constraints of each sublayer and solving the resulting system of constraints for the frame of each sublayer starting from the bounds of the containing layer.

Sibling layers are referenced by name, using the `name` property of each layer. The special name `superlayer` is used to refer to the layer's superlayer.

**Important:** It is possible to specify a set of constraints for a layer (for example, circular attribute dependencies) that will cause layout to fail. In that case the behavior is undefined.

## Tasks

### Creating the Layout Manager

+ [LayoutManager](#) (page 40)

Creates and returns a new `CAConstraintLayoutManager` instance.

## Class Methods

### `layoutManager`

Creates and returns a new `CAConstraintLayoutManager` instance.

+ (id)layoutManager

#### **Return Value**

A new `CAConstraintLayoutManager` instance.

#### **Availability**

Available in Mac OS X v10.5 and later.

#### **Related Sample Code**

Core Animation QuickTime Layer

CoreAnimationKioskStyleMenu

#### **Declared In**

`CAConstraintLayoutManager.h`



# CAEmitterCell Class Reference

---

<b>Inherits from</b>	NSObject
<b>Conforms to</b>	NSCoding CAMediaTiming NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.6 and later.
<b>Declared in</b>	CAEmitterCell.h
<b>Companion guides</b>	Core Animation Programming Guide Core Animation Cookbook
<b>Related sample code</b>	Fire Fireworks

## Overview

The `CAEmitterCell` class represents one source of particles being emitted in a `CAEmitterLayer`. An emitter cell define the direction in which they emit and the properties of the emitted particles. Emitter cells can have an array of sub-cells, which lets the particles themselves emit particles.

## Tasks

### Creating and Initializing an Emitter Cell

+ [emitterCell](#) (page 58)  
Creates and returns an instance of `CAEmitterCell`.

### Providing Emitter Cell Content

[contents](#) (page 46) *property*  
An object that provides the contents of the layer. Animatable.

[contentsRect](#) (page 47) *property*

A rectangle, in the unit coordinate space, defining the subrectangle of [contents](#) (page 46) that the receiver should draw. Animatable.

[emitterCells](#) (page 49) *property*

An optional array containing the sub-cells of this cell.

## Setting Emitter Cell Visual Attributes

[enabled](#) (page 49) *property*

Determines whether or not cells from this emitter are rendered.

[color](#) (page 46) *property*

Determines the color of each emitted object. Animatable.

[redRange](#) (page 52) *property*

Specifies the range the red color component of the cell can vary. Animatable.

[greenRange](#) (page 49) *property*

Specifies the range the green color component of the cell can vary. Animatable.

[blueRange](#) (page 45) *property*

Specifies the range the blue color component of the cell can vary. Animatable.

[alphaRange](#) (page 44) *property*

Specifies the range the alpha color component of the cell can vary. Animatable.

[redSpeed](#) (page 53) *property*

The speed, in seconds, at which the red color component changes over the lifetime of the cell. Animatable.

[greenSpeed](#) (page 50) *property*

The speed, in seconds, at which the green color component changes over the lifetime of the cell. Animatable.

[blueSpeed](#) (page 45) *property*

The speed, in seconds, at which the blue color component changes over the lifetime of the cell. Animatable.

[alphaSpeed](#) (page 44) *property*

The speed, in seconds, at which the alpha color component changes over the lifetime of the cell. Animatable.

[magnificationFilter](#) (page 51) *property*

The filter used when increasing the size of the content.

[minificationFilter](#) (page 51) *property*

The filter used when reducing the size of the content.

[minificationFilterBias](#) (page 51) *property*

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

[scale](#) (page 53) *property*

Specifies the scale factor applied to the cell. Animatable.

[scaleRange](#) (page 54) *property*

Specifies the range the blue color component of the cell can vary. Animatable.

[name](#) (page 52) *property*

The name of the cell.

[style](#) (page 55) *property*

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

## Emitter Cell Motion Attributes

[spin](#) (page 54) *property*

Specifies the rotation velocity per second, in radians, applied to the cell. Animatable.

[spinRange](#) (page 55) *property*

Specifies the range the spin of the cell can vary over its lifetime. Animatable.

[emissionLatitude](#) (page 47) *property*

Defines the latitudinal orientation of the emission angle. Animatable.

[emissionLongitude](#) (page 48) *property*

Defines the longitudinal orientation of the emission angle. Animatable.

[emissionRange](#) (page 48) *property*

An angle, in radians, defining a cone around the emission angle. Animatable.

## Emission Cell Temporal Attributes

[lifetime](#) (page 50) *property*

The lifetime of the cell, in seconds. Animatable.

[lifetimeRange](#) (page 51) *property*

Specifies the mean value the [lifetime](#) (page 50) of the cell can vary. Animatable.

[birthRate](#) (page 45) *property*

Number of emitted objects created every second. Animatable.

[scaleSpeed](#) (page 54) *property*

The speed, in seconds, at which the scale changes over the lifetime of the cell. Animatable.

[velocity](#) (page 56) *property*

Specifies the initial velocity of the cell. Animatable.

[velocityRange](#) (page 56) *property*

Specifies the range the velocity of the cell can vary. Animatable.

[xAcceleration](#) (page 56) *property*

Specifies the x component of an acceleration vector applied to cell.

[yAcceleration](#) (page 57) *property*

Specifies the y component of an acceleration vector applied to cell.

[zAcceleration](#) (page 57) *property*

Specifies the z component of an acceleration vector applied to cell.

## Key-Value Coding Extensions

+ [defaultValueForKey:](#) (page 57)

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

- [shouldArchiveValueForKey:](#) (page 58)

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

## Properties

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

### alphaRange

Specifies the range the alpha color component of the cell can vary. Animatable.

```
@property float alphaRange
```

#### Discussion

The range specifies the mean amount the alpha component of the [color](#) (page 46) property can vary for the cell.

Default is 0.0.

#### Availability

Available in Mac OS X v10.6 and later.

#### See Also

[@property color](#) (page 46)

[@property alphaSpeed](#) (page 44)

#### Declared In

CAEmitterCell.h

### alphaSpeed

The speed, in seconds, at which the alpha color component changes over the lifetime of the cell. Animatable.

```
@property float alphaSpeed
```

#### Discussion

The speed change is defined as the rate of change per second.

Defaults to 0.

#### Availability

Available in Mac OS X v10.6 and later.

#### See Also

[@property color](#) (page 46)

[@property alphaRange](#) (page 44)

#### Related Sample Code

Fire

Fireworks

#### Declared In

CAEmitterCell.h

## birthRate

Number of emitted objects created every second. Animatable.

```
@property float birthRate
```

### Discussion

Default value is 0.

### Availability

Available in Mac OS X v10.6 and later.

### Related Sample Code

Fire

Fireworks

### Declared In

CAEmitterCell.h

## blueRange

Specifies the range the blue color component of the cell can vary. Animatable.

```
@property float blueRange
```

### Discussion

The range specifies the mean amount the blue component of the [color](#) (page 46) property can vary for the cell.

Default value is 0.0.

### Availability

Available in Mac OS X v10.6 and later.

### See Also

[@property color](#) (page 46)

[@property blueSpeed](#) (page 45)

### Related Sample Code

Fireworks

### Declared In

CAEmitterCell.h

## blueSpeed

The speed, in seconds, at which the green color component changes over the lifetime of the cell. Animatable.

```
@property float blueSpeed
```

### Discussion

The speed change is defined as the rate of change per second.

Defaults to 0.

**Availability**

Available in Mac OS X v10.6 and later.

**See Also**

[@property color](#) (page 46)

[@property blueRange](#) (page 45)

**Related Sample Code**

Fireworks

**Declared In**

CAEmitterCell.h

**color**

Determines the color of each emitted object. Animatable.

`@property CGColorRef color`

**Discussion**

The specified color of the cell will vary by a random amount within the [redRange](#) (page 52), [greenRange](#) (page 49), [blueRange](#) (page 45) and [alphaRange](#) (page 44) values over the lifetime of the cell. The [redSpeed](#) (page 53), [greenSpeed](#) (page 50), [blueSpeed](#) (page 45), and [alphaSpeed](#) (page 44) determine the rate of change.

Defaults to opaque white.

**Availability**

Available in Mac OS X v10.6 and later.

**See Also**

[@property color](#) (page 46)

[@property redRange](#) (page 52)

[@property greenRange](#) (page 49)

[@property alphaRange](#) (page 44)

[@property blueRange](#) (page 45)

[@property redSpeed](#) (page 53)

[@property greenSpeed](#) (page 50)

[@property blueSpeed](#) (page 45)

[@property alphaSpeed](#) (page 44)

**Related Sample Code**

Fire

Fireworks

**Declared In**

CAEmitterCell.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.

Default value is `nil`.

**Availability**

Available in Mac OS X v10.6 and later.

**See Also**

[@property contentsRect](#) (page 47)

**Related Sample Code**

Fire

Fireworks

**Declared In**

CAEmitterCell.h

## contentsRect

A rectangle, in the unit coordinate space, defining the subrectangle of [contents](#) (page 46) 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 Mac OS X v10.6 and later.

**See Also**

[@property contents](#) (page 46)

**Declared In**

CAEmitterCell.h

## emissionLatitude

Defines the latitudinal orientation of the emission angle. Animatable.

```
@property CGFloat emissionLatitude
```

**Discussion**

The emission latitude is the orientation of the emission angle from the z-axis. It is also referred to as the colatitude.

Default is 0.0.

**Availability**

Available in Mac OS X v10.6 and later.

**Related Sample Code**

Fireworks

**Declared In**

CAEmitterCell.h

## emissionLongitude

Defines the longitudinal orientation of the emission angle. Animatable.

```
@property CGFloat emissionLongitude
```

**Discussion**

The emission longitude is the orientation of the emission angle in the xy-plane. It is also often referred to as the azimuth.

Default is 0.0.

**Availability**

Available in Mac OS X v10.6 and later.

**Related Sample Code**

Fire

Fireworks

**Declared In**

CAEmitterCell.h

## emissionRange

An angle, in radians, defining a cone around the emission angle. Animatable.

```
@property CGFloat emissionRange
```

**Discussion**

Cells are uniformly distributed across this cone.

Defaults to 0.

**Availability**

Available in Mac OS X v10.6 and later.

**Related Sample Code**

Fire

Fireworks

**Declared In**

CAEmitterCell.h



## emitterCells

An optional array containing the sub-cells of this cell.

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

### Discussion

When specified, each particle emitted by the cell will act as an emitter for each of the cell's sub-cells. The emission point is the current particle position and the emission angle is relative to the current direction of the particle.

Default is `nil`.

### Availability

Available in Mac OS X v10.6 and later.

### Related Sample Code

Fireworks

### Declared In

CAEmitterCell.h

## enabled

Determines whether or not cells from this emitter are rendered.

```
@property(getter=isEnabled) BOOL enabled
```

### Discussion

Default is `YES`.

### Availability

Available in Mac OS X v10.6 and later.

### Declared In

CAEmitterCell.h

## greenRange

Specifies the range the green color component of the cell can vary. Animatable.

```
@property float greenRange
```

### Discussion

The range specifies the mean amount the green component of the [color](#) (page 46) property can vary for the cell.

Default is `0.0`.

### Availability

Available in Mac OS X v10.6 and later.

### See Also

[@property color](#) (page 46)

[@property greenSpeed](#) (page 50)

**Related Sample Code**

Fireworks

**Declared In**

CAEmitterCell.h

## greenSpeed

The speed, in seconds, at which the green color component changes over the lifetime of the cell. Animatable.

```
@property float greenSpeed
```

**Discussion**

The speed change is defined as the rate of change per second.

Defaults to 0.

**Availability**

Available in Mac OS X v10.6 and later.

**Related Sample Code**

Fireworks

**Declared In**

CAEmitterCell.h

## lifetime

The lifetime of the cell, in seconds. Animatable.

```
@property float lifetime
```

**Discussion**

The lifetime of the cell will vary by a random amount with the range specified by [lifetimeRange](#) (page 51).

Default is 0.0.

**Availability**

Available in Mac OS X v10.6 and later.

**Related Sample Code**

Fire

Fireworks

**Declared In**

CAEmitterCell.h

## lifetimeRange

Specifies the mean value the [lifetime](#) (page 50) of the cell can vary. Animatable.

```
@property float lifetimeRange
```

### Discussion

If the [lifetimeRange](#) (page 51) is 3 seconds, and the [lifetime](#) (page 50) of the cell is 10 seconds, the cell's actual lifetime will be between 7 and 13 seconds.

Default is 0.0.

### Availability

Available in Mac OS X v10.6 and later.

### Declared In

CAEmitterCell.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 132). The default value is [kCAFilterLinear](#) (page 133).

### Availability

Available in Mac OS X v10.6 and later.

### Declared In

CAEmitterCell.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 132). The default value is [kCAFilterLinear](#) (page 133).

### Availability

Available in Mac OS X v10.6 and later.

### Declared In

CAEmitterCell.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 51) when it is set to `kCAFilterTrilinear`.

Defaults to 0.

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAEmitterCell.h

**name**

The name of the cell.

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

**Discussion**

The cell name is used when constructing animation key paths that reference the cell. Defaults to `nil`.

For example, adding an animation to a cell's enclosing layer with the a keypath such as `emitterCells.myCellName.redRange` would animate the `redRange` property of the cell in the layer's `emitterCells` array with the name `myCellName`.

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAEmitterCell.h

**redRange**

Specifies the range the red color component of the cell can vary. Animatable.

```
@property float redRange
```

**Discussion**

The range specifies the mean amount the red component of the [color](#) (page 46) property can vary for the cell.

Default is 0.0.

**Availability**

Available in Mac OS X v10.6 and later.

**See Also**

[@property color](#) (page 46)

[@property redSpeed](#) (page 53)

**Related Sample Code**

Fireworks

**Declared In**

CAEmitterCell.h

## redSpeed

The speed, in seconds, at which the red color component changes over the lifetime of the cell. Animatable.

```
@property float redSpeed
```

**Discussion**

The speed change is defined as the rate of change per second.

Defaults to 0.

**Availability**

Available in Mac OS X v10.6 and later.

**See Also**

[@property color](#) (page 46)

[@property redRange](#) (page 52)

**Related Sample Code**

Fireworks

**Declared In**

CAEmitterCell.h

## scale

Specifies the scale factor applied to the cell. Animatable.

```
@property CGFloat scale
```

**Discussion**

The scale of the cell will vary by a random amount with the range specified by [scaleRange](#) (page 54). The [scaleSpeed](#) (page 54) property determines the rate of change.

Default is 0.0.

**Availability**

Available in Mac OS X v10.6 and later.

**See Also**

[@property scaleRange](#) (page 54)

[@property scaleSpeed](#) (page 54)

**Related Sample Code**

Fire

Fireworks

**Declared In**

CAEmitterCell.h

## scaleRange

Specifies the range the blue color component of the cell can vary. Animatable.

```
@property CGFloat scaleRange
```

### Discussion

The range specifies the mean amount the [scale](#) (page 53) value can vary for the cell over its lifetime. The [scaleSpeed](#) (page 54) property determines the rate of change.

### Availability

Available in Mac OS X v10.6 and later.

### See Also

[@property scale](#) (page 53)

[@property scaleSpeed](#) (page 54)

### Related Sample Code

Fireworks

### Declared In

CAEmitterCell.h

## scaleSpeed

The speed, in seconds, at which the scale changes over the lifetime of the cell. Animatable.

```
@property CGFloat scaleSpeed
```

### Discussion

The speed change is defined as the rate of change per second.

Defaults to 0.

### Availability

Available in Mac OS X v10.6 and later.

### Related Sample Code

Fire

Fireworks

### Declared In

CAEmitterCell.h

## spin

Specifies the rotation velocity per second, in radians, applied to the cell. Animatable.

```
@property CGFloat spin
```

### Discussion

The spin of the cell will vary by a random amount with the range specified by [spinRange](#) (page 55).

Default is 0.0.

**Availability**

Available in Mac OS X v10.6 and later.

**Related Sample Code**

Fire

Fireworks

**Declared In**

CAEmitterCell.h

## spinRange

Specifies the range the spin of the cell can vary over its lifetime. Animatable.

```
@property CGFloat spinRange
```

**Discussion**

The range specifies the mean amount the [spin](#) (page 54) value can vary over the cell's lifetime.

Default value is 0.0.

**Availability**

Available in Mac OS X v10.6 and later.

**Related Sample Code**

Fire

**Declared In**

CAEmitterCell.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 cell's [defaultValueForKey:](#) (page 57) class 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 cell's properties are undefined until the `style` property is reset.

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAEmitterCell.h

**velocity**

Specifies the initial velocity of the cell. Animatable.

```
@property CGFloat velocity
```

**Discussion**

The velocity of the cell will vary by a random amount within the range specified by [velocityRange](#) (page 56).

**Availability**

Available in Mac OS X v10.6 and later.

**Related Sample Code**

Fire

Fireworks

**Declared In**

CAEmitterCell.h

**velocityRange**

Specifies the range the velocity of the cell can vary. Animatable.

```
@property CGFloat velocityRange
```

**Discussion**

The range specifies the mean amount the initial [velocity](#) (page 56) value change.

Default is 0.0.

**Availability**

Available in Mac OS X v10.6 and later.

**Related Sample Code**

Fire

Fireworks

**Declared In**

CAEmitterCell.h

**xAcceleration**

Specifies the x component of an acceleration vector applied to cell.

```
@property CGFloat xAcceleration
```

**Discussion**

Default is 0.0.



**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAEmitterCell.h

## yAcceleration

Specifies the y component of an acceleration vector applied to cell.

```
@property CGFloat yAcceleration
```

**Discussion**

Default is 0.0.

**Availability**

Available in Mac OS X v10.6 and later.

**Related Sample Code**

Fire

Fireworks

**Declared In**

CAEmitterCell.h

## zAcceleration

Specifies the z component of an acceleration vector applied to cell.

```
@property CGFloat zAcceleration
```

**Discussion**

Default is 0.0.

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAEmitterCell.h

## Class Methods

**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 Mac OS X v10.6 and later.

**Declared In**

`CAEmitterCell.h`

**emitterCell**

Creates and returns an instance of `CAEmitterCell`.

```
+ (id)emitterCell
```

**Return Value**

The initialized `CAEmitterCell` object or `nil` if initialization is not successful.

**Availability**

Available in Mac OS X v10.6 and later.

**Related Sample Code**

Fire

Fireworks

**Declared In**

`CAEmitterCell.h`

## Instance Methods

**shouldArchiveValueForKey:**

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

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

**Parameters**

*key*

The name of one of the receiver's properties.

**Return Value**

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

**Discussion**

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

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAEmitterCell.h



# CAEmitterLayer Class Reference

---

<b>Inherits from</b>	CALayer : NSObject
<b>Conforms to</b>	NSCoding (CALayer) CAMediaTiming (CALayer) NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.6 and later.
<b>Declared in</b>	CAEmitterLayer.h
<b>Companion guides</b>	Core Animation Programming Guide Core Animation Cookbook
<b>Related sample code</b>	Fire Fireworks

## Overview

The `CAEmitterLayer` class provides a particle emitter system for Core Animation. The particles are defined by instances of `CAEmitterCell`.

The particles are drawn above the layer's background color and border

## Tasks

### Specifying Particle Emitter Cells

[emitterCells](#) (page 63) *property*

The array emitter cells attached to the layer.

### Emitter Geometry

[renderMode](#) (page 66) *property*

Defines how particle cells are rendered into the layer.

[emitterPosition](#) (page 64) *property*

The position of the center of the particle emitter. Animatable.

[emitterShape](#) (page 64) *property*

Specifies the emitter shape.

[emitterZPosition](#) (page 65) *property*

Specifies the center of the particle emitter shape along the z-axis. Animatable.

[emitterDepth](#) (page 63) *property*

Determines the depth of the emitter shape.

[emitterSize](#) (page 65) *property*

Determines the size of the particle emitter shape. Animatable.

## Emitter Cell Attribute Multipliers

[scale](#) (page 66) *property*

Defines a multiplier applied to the cell-defined particle scale.

[seed](#) (page 67) *property*

Specifies the seed used to initialize the random number generator.

[spin](#) (page 67) *property*

Defines a multiplier applied to the cell-defined particle spin. Animatable.

[velocity](#) (page 67) *property*

Defines a multiplier applied to the cell-defined particle velocity. Animatable.

[birthRate](#) (page 62) *property*

Defines a multiplier that is applied to the cell-defined birth rate. Animatable

[emitterMode](#) (page 63) *property*

Specifies the emitter mode.

[lifetime](#) (page 65) *property*

Defines a multiplier applied to the cell-defined lifetime range when particles are created. Animatable.

[preservesDepth](#) (page 66) *property*

Defines whether the layer flattens its the particles into its plane.

## Properties

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

### birthRate

Defines a multiplier that is applied to the cell-defined birth rate. Animatable

```
@property float birthRate
```

#### Discussion

The birth rate of each cell is multiplied by this number to give the actual number of particles created every second. Default value is 1.0.

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAEmitterLayer.h

**emitterCells**

The array emitter cells attached to the layer.

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

**Discussion**

Each object in the array must be an instance of the `CAEmitterCell` class.

**Availability**

Available in Mac OS X v10.6 and later.

**Related Sample Code**

Fire

Fireworks

**Declared In**

CAEmitterLayer.h

**emitterDepth**

Determines the depth of the emitter shape.

```
@property CGFloat emitterDepth
```

**Discussion**

How the emitter depth is applied depends on the emitter shape. See [“Emitter Shape”](#) (page 68) for details. Depending on the value of `emitterShape` (page 64), this value may be ignored.

Default is 0.0.

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAEmitterLayer.h

**emitterMode**

Specifies the emitter mode.

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

**Discussion**

The possible values for `emitterMode` are shown in [“Emitter Modes”](#) (page 69). The default value is `kCAEmitterLayerVolume` (page 69).

**Availability**

Available in Mac OS X v10.6 and later.

**Related Sample Code**

Fire

**Declared In**

CAEmitterLayer.h

**emitterPosition**

The position of the center of the particle emitter. Animatable.

```
@property CGPoint emitterPosition
```

**Discussion**

See “[Emitter Shape](#)” (page 68) for details of how the `emitterPosition` relates to the possible emitter shapes.

Default is (0.0,0.0).

**Availability**

Available in Mac OS X v10.6 and later.

**Related Sample Code**

Fire

Fireworks

**Declared In**

CAEmitterLayer.h

**emitterShape**

Specifies the emitter shape.

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

**Discussion**

The possible values for `emitterMode` are shown in “[Emitter Shape](#)” (page 68). The default value is `kCAEmitterLayerPoint` (page 68).

**Availability**

Available in Mac OS X v10.6 and later.

**Related Sample Code**

Fire

**Declared In**

CAEmitterLayer.h



## emitterSize

Determines the size of the particle emitter shape. Animatable.

```
@property CGSize emitterSize
```

### Discussion

How the emitter size is applied depends on the emitter shape. See “[Emitter Shape](#)” (page 68) for details. Depending on the value of `emitterShape` (page 64), this value may be ignored.

Default is 0.0.

### Availability

Available in Mac OS X v10.6 and later.

### Related Sample Code

Fire

### Declared In

CAEmitterLayer.h

## emitterZPosition

Specifies the center of the particle emitter shape along the z-axis. Animatable.

```
@property CGFloat emitterZPosition
```

### Discussion

See “[Emitter Shape](#)” (page 68) for details of how the emitterZPosition relates to the possible emitter shapes.

Default is 0.0.

### Availability

Available in Mac OS X v10.6 and later.

### Declared In

CAEmitterLayer.h

## lifetime

Defines a multiplier applied to the cell-defined lifetime range when particles are created. Animatable.

```
@property float lifetime
```

### Discussion

Default value is 1.0.

### Availability

Available in Mac OS X v10.6 and later.

### Declared In

CAEmitterLayer.h

## preservesDepth

Defines whether the layer flattens its the particles into its plane.

```
@property BOOL preservesDepth
```

### Discussion

If YES, the layer renders its particles as if they directly inhabit the three-dimensional coordinate space of the layer's superlayer. When enabled, the effect of the layer's `filters`, `backgroundFilters`, and shadow related properties is undefined.

Default is NO.

### Availability

Available in Mac OS X v10.6 and later.

### Declared In

CAEmitterLayer.h

## renderMode

Defines how particle cells are rendered into the layer.

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

### Discussion

The possible values for render modes are shown in “[Emitter Modes](#)” (page 69). The default value is `kCAEmitterLayerUnordered` (page 70).

### Availability

Available in Mac OS X v10.6 and later.

### Related Sample Code

Fire

Fireworks

### Declared In

CAEmitterLayer.h

## scale

Defines a multiplier applied to the cell-defined particle scale.

```
@property float scale
```

### Discussion

Default value is 1.0.

### Availability

Available in Mac OS X v10.6 and later.

### Declared In

CAEmitterLayer.h

## seed

Specifies the seed used to initialize the random number generator.

```
@property unsigned int seed
```

### Discussion

Each layer has its own random number generator state. Emitter cell properties that are defined as a mean and a range, such as a cell's `speed`, the value of the properties are uniformly distributed in the interval  $[M - R/2, M + R/2]$ .

### Availability

Available in Mac OS X v10.6 and later.

### Declared In

CAEmitterLayer.h

## spin

Defines a multiplier applied to the cell-defined particle spin. Animatable.

```
@property float spin
```

### Discussion

Default value is 1.0.

### Availability

Available in Mac OS X v10.6 and later.

### Declared In

CAEmitterLayer.h

## velocity

Defines a multiplier applied to the cell-defined particle velocity. Animatable.

```
@property float velocity
```

### Discussion

Default value is 1.0.

### Availability

Available in Mac OS X v10.6 and later.

### Declared In

CAEmitterLayer.h

## Constants

### Emitter Shape

The emission shape is a one, two or three dimensional shape that defines where the emitted particles originate. The shapes are defined by a subset of [emitterPosition](#) (page 64), [emitterZPosition](#) (page 65), [emitterSize](#) (page 65) and [emitterDepth](#) (page 63) properties.

```
NSString * const kCAEmitterLayerPoint;
NSString * const kCAEmitterLayerLine;
NSString * const kCAEmitterLayerRectangle;
NSString * const kCAEmitterLayerCuboid;
NSString * const kCAEmitterLayerCircle;
NSString * const kCAEmitterLayerSphere;
```

#### Constants

`kCAEmitterLayerPoint`

Particles are emitted from a single point at (`emitterPosition.x`, `emitterPosition.y`, `emitterZPosition`)

Available in Mac OS X v10.6 and later.

Declared in `CAEmitterLayer.h`.

`kCAEmitterLayerLine`

Particles are emitted along a line from (`emitterPosition.x - emitterSize.width/2`, `emitterPosition.y`, `emitterZPosition`) to (`emitterPosition.x + emitterSize.width/2`, `emitterPosition.y`, `emitterZPosition`).

Available in Mac OS X v10.6 and later.

Declared in `CAEmitterLayer.h`.

`kCAEmitterLayerRectangle`

Particles are emitted from a rectangle with opposite corners [`emitterPosition.x - emitterSize.width/2`, `emitterPosition.y - emitterSize.height/2`, `emitterZPosition`], [`emitterPosition.x + emitterSize.width/2`, `emitterPosition.y + emitterSize.height/2`, `emitterZPosition`].

Available in Mac OS X v10.6 and later.

Declared in `CAEmitterLayer.h`.

`kCAEmitterLayerCuboid`

Particles are emitted from a cuboid (3D rectangle) with opposite corners: [`emitterPosition.x - emitterSize.width/2`, `emitterPosition.y - emitterSize.height/2`, `emitterZPosition - emitterDepth/2`], [`emitterPosition.x + emitterSize.width/2`, `emitterPosition.y + emitterSize.height/2`, `emitterZPosition+emitterDepth/2`].

Available in Mac OS X v10.6 and later.

Declared in `CAEmitterLayer.h`.

`kCAEmitterLayerCircle`

Particles are emitted from a circle centered at (`emitterPosition.x`, `emitterPosition.y`, `emitterZPosition`) of radius `emitterSize.width`.

Available in Mac OS X v10.6 and later.

Declared in `CAEmitterLayer.h`.

kCAEmitterLayerSphere

Particles are emitted from a sphere centered at (emitterPosition.x, emitterPosition.y, emitterZPosition) of radius emitterSize.width.

Available in Mac OS X v10.6 and later.

Declared in CAEmitterLayer.h.

## Emitter Modes

These constants specify the possible emitter modes. They are used by the [emitterMode](#) (page 63) property.

```
NSString * const kCAEmitterLayerPoints;
NSString * const kCAEmitterLayerOutline;
NSString * const kCAEmitterLayerSurface;
NSString * const kCAEmitterLayerVolume;
```

### Constants

kCAEmitterLayerPoints

Particles are emitted from points on the particle emitter.

Available in Mac OS X v10.6 and later.

Declared in CAEmitterLayer.h.

kCAEmitterLayerOutline

Particles are emitted from the outline of the particle emitter.

Available in Mac OS X v10.6 and later.

Declared in CAEmitterLayer.h.

kCAEmitterLayerSurface

Particles are emitted from the surface of the particle emitter.

Available in Mac OS X v10.6 and later.

Declared in CAEmitterLayer.h.

kCAEmitterLayerVolume

Particles are emitted from the a position within the particle emitter.

Available in Mac OS X v10.6 and later.

Declared in CAEmitterLayer.h.

## Emitter Render Order

These constants specify the order that emitter cells are composited. They are used by the [renderMode](#) (page 66) property.

```
NSString * const kCAEmitterLayerUnordered;  
NSString * const kCAEmitterLayerOldestFirst;  
NSString * const kCAEmitterLayerOldestLast;  
NSString * const kCAEmitterLayerBackToFront;  
NSString * const kCAEmitterLayerAdditive;
```

**Constants**

`kCAEmitterLayerUnordered`

Particles are rendered unordered. This mode uses source-over compositing.

Available in Mac OS X v10.6 and later.

Declared in `CAEmitterLayer.h`.

`kCAEmitterLayerOldestFirst`

Particles are rendered oldest first. This mode uses source-over compositing.

Available in Mac OS X v10.6 and later.

Declared in `CAEmitterLayer.h`.

`kCAEmitterLayerOldestLast`

Particles are rendered oldest last. This mode uses source-over compositing.

Available in Mac OS X v10.6 and later.

Declared in `CAEmitterLayer.h`.

`kCAEmitterLayerBackToFront`

Particles are rendered from back to front, sorted by z-position. This mode uses source-over compositing.

Available in Mac OS X v10.6 and later.

Declared in `CAEmitterLayer.h`.

`kCAEmitterLayerAdditive`

The particles are rendered using source-additive compositing.

Available in Mac OS X v10.6 and later.

Declared in `CAEmitterLayer.h`.

# CAGradientLayer Class Reference

---

<b>Inherits from</b>	CALayer : NSObject
<b>Conforms to</b>	NSCoding (CALayer) CAMediaTiming (CALayer) NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.6 and later.
<b>Declared in</b>	CAGradientLayer.h

## Overview

The `CAGradientLayer` class draws a color gradient over its background color, filling the shape of the layer (including rounded corners)

## Tasks

### Gradient Style Properties

`colors` (page 72) *property*

An array of `CGColorRef` objects defining the color of each gradient stop. Animatable.

`locations` (page 72) *property*

An optional array of `NSNumber` objects defining the location of each gradient stop. Animatable.

`endPoint` (page 72) *property*

The end point of the gradient when drawn in the layer's coordinate space. Animatable.

`startPoint` (page 73) *property*

The start point of the gradient when drawn in the layer's coordinate space. Animatable.

`type` (page 73) *property*

Style of gradient drawn by the layer.

## Properties

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

## colors

An array of `CGColorRef` objects defining the color of each gradient stop. Animatable.

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

### Discussion

Defaults to `nil`.

### Availability

Available in Mac OS X v10.6 and later.

### Declared In

`CAGradientLayer.h`

## endPoint

The end point of the gradient when drawn in the layer's coordinate space. Animatable.

```
@property CGPoint endPoint
```

### Discussion

The end point corresponds to the last stop of the gradient. The point is defined in the unit coordinate space and is then mapped to the layer's bounds rectangle when drawn.

Default value is `(0.5, 1.0)`.

### Availability

Available in Mac OS X v10.6 and later.

### Declared In

`CAGradientLayer.h`

## locations

An optional array of `NSNumber` objects defining the location of each gradient stop. Animatable.

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

### Discussion

The gradient stops are specified as values between 0 and 1. The values must be monotonically increasing. If `nil`, the stops are spread uniformly across the range. Defaults to `nil`.

When rendered, the colors are mapped to the output color space before being interpolated.

### Availability

Available in Mac OS X v10.6 and later.

### Declared In

`CAGradientLayer.h`



## startPoint

The start point of the gradient when drawn in the layer’s coordinate space. Animatable.

```
@property CGPoint startPoint
```

### Discussion

The start point corresponds to the first stop of the gradient. The point is defined in the unit coordinate space and is then mapped to the layer’s bounds rectangle when drawn.

Default value is (0.5,0.0).

### Availability

Available in Mac OS X v10.6 and later.

### Declared In

CAGradientLayer.h

## type

Style of gradient drawn by the layer.

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

### Discussion

Defaults to [kCAGradientLayerAxial](#) (page 73).

### Availability

Available in Mac OS X v10.6 and later.

### Declared In

CAGradientLayer.h

## Constants

### Gradient Types

The style of gradient drawn by the layer.

```
NSString * const kCAGradientLayerAxial
```

### Constants

```
kCAGradientLayerAxial
```

An axial gradient (also called a linear gradient) varies along an axis between two defined end points. All points that lie on a line perpendicular to the axis have the same color value.

Available in Mac OS X v10.6 and later.

Declared in CAGradientLayer.h.



# CAKeyframeAnimation Class Reference

---

<b>Inherits from</b>	CAPropertyAnimation : CAAnimation : NSObject
<b>Conforms to</b>	NSCoding (CAAnimation) NSCopying (CAAnimation) CAAction (CAAnimation) CAMediaTiming (CAAnimation) NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.5 and later.
<b>Declared in</b>	CAAnimation.h
<b>Companion guides</b>	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 149) 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 77) *property*

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

`values` (page 78) *property*

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

## Keyframe Timing

[keyTimes](#) (page 76) *property*

An optional array of `NSNumber` objects that define the duration of each keyframe segment.

[timingFunctions](#) (page 78) *property*

An optional array of `CAMediaTimingFunction` instances that defines the pacing of the each keyframe segment.

[calculationMode](#) (page 76) *property*

Specifies how intermediate keyframe values are calculated by the receiver.

## Rotation Mode Attribute

[rotationMode](#) (page 77) *property*

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

## Properties

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

### 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 79). The default is `kCAAnimationLinear` (page 79).

#### Availability

Available in Mac OS X v10.5 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 76) 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 Mac OS X v10.5 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 78) 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 76) should be set to `kCAAnimationPaced` (page 79).

#### Availability

Available in Mac OS X v10.5 and later.

#### See Also

[@property rotationMode](#) (page 77)

#### 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 78). 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 Mac OS X v10.5 and later.

**See Also**

[@property path](#) (page 77)

**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 Mac OS X v10.5 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 77) property is used.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CAAnimation.h

## Constants

### Rotation Mode Values

These constants are used by the [rotationMode](#) (page 77) property.

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

**Constants**

```
kCAAnimationRotateAuto
```

The objects travel on a tangent to the path.

Available in Mac OS X v10.5 and later.

Declared in `CAAnimation.h`.

```
kCAAnimationRotateAutoReverse
```

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

Available in Mac OS X v10.5 and later.

Declared in `CAAnimation.h`.

**Value calculation modes**

These constants are used by the `calculationMode` (page 76) property.

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

**Constants**

```
kCAAnimationLinear
```

Simple linear calculation between keyframe values.

Available in Mac OS X v10.5 and later.

Declared in `CAAnimation.h`.

```
kCAAnimationDiscrete
```

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

Available in Mac OS X v10.5 and later.

Declared in `CAAnimation.h`.

```
kCAAnimationPaced
```

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

Available in Mac OS X v10.5 and later.

Declared in `CAAnimation.h`.





# CALayer Class Reference

---

<b>Inherits from</b>	NSObject
<b>Conforms to</b>	NSCoding CAMediaTiming NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.5 and later.
<b>Declared in</b>	CAConstraintLayoutManager.h CALayer.h CAScrollLayer.h
<b>Companion guides</b>	Core Animation Programming Guide Core Animation Cookbook
<b>Related sample code</b>	CoreAnimationKioskStyleMenu GeekGameBoard ImageBrowserViewAppearance ImageKitDemo LightTable

## 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 107)  
Creates and returns an instance of `CALayer`.
- [init](#) (page 116)  
Returns an initialized `CALayer` object.

- [initWithLayer:](#) (page 117)  
Override to copy or initialize custom fields of the specified layer.

## Accessing the Presentation Layer

- [presentationLayer](#) (page 121)  
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 120)  
Returns the model layer of the receiver, if it represents a current presentation layer.

## Modifying the Layer Geometry

- [frame](#) (page 96) *property*  
Specifies receiver's frame rectangle in the super-layer's coordinate space.
- [bounds](#) (page 91) *property*  
Specifies the bounds rectangle of the receiver. Animatable.
- [position](#) (page 101) *property*  
Specifies the receiver's position in the superlayer's coordinate system. Animatable.
- [zPosition](#) (page 106) *property*  
Specifies the receiver's position on the z axis. Animatable.
- [anchorPointZ](#) (page 88) *property*  
The Z component of the layer's anchor point. Animatable.
- [anchorPoint](#) (page 88) *property*  
Defines the anchor point of the layer's bounds rectangle. Animatable.
- [affineTransform](#) (page 110)  
Convenience method for getting the [transform](#) (page 105) property as an affine transform.
- [setAffineTransform:](#) (page 125)  
Convenience method for setting the [transform](#) (page 105) property as an affine transform.
- [transform](#) (page 105) *property*  
Specifies the transform applied to the receiver, relative to the center of its bounds. Animatable.
- [sublayerTransform](#) (page 104) *property*  
Specifies a transform applied to each sublayer when rendering. Animatable.

## Providing Layer Content

- [contents](#) (page 92) *property*  
An object that provides the contents of the layer. Animatable.
- [contentsRect](#) (page 94) *property*  
A rectangle, in the unit coordinate space, defining the subrectangle of [contents](#) (page 92) that the receiver should draw. Animatable.
- [contentsCenter](#) (page 93) *property*  
Specifies the area of the content image that should be scaled. Animatable.

- `display` (page 115)  
Reload the content of this layer.
- `displayLayer:` (page 127) *delegate method*  
Allows the delegate to override the `display` (page 115) implementation.
- `drawInContext:` (page 116)  
Draws the receiver's content in the specified graphics context.
- `drawLayer:inContext:` (page 128) *delegate method*  
Allows the delegate to override the layer's `drawInContext:` implementation.
- `opaque` (page 101) *property*  
This property affects how the layer creates the content directly.
- `edgeAntialiasingMask` (page 95) *property*  
A bitmask defining how the edges of the receiver are rasterized.
- `contentsAreFlipped` (page 112)  
Returns whether the layer content is implicitly flipped when rendered.
- `geometryFlipped` (page 97) *property*  
Determines if the geometry of the layer and its sublayers are flipped vertically.

## Style Attributes

- `contentsGravity` (page 93) *property*  
Determines how the receiver's contents are positioned within its bounds.
- `opacity` (page 100) *property*  
Determines the opacity of the receiver. Animatable.
- `hidden` (page 97) *property*  
Determines whether the receiver is displayed. Animatable.
- `masksToBounds` (page 99) *property*  
Determines if the sublayers are clipped to the receiver's bounds. Animatable.
- `doubleSided` (page 95) *property*  
Determines whether the receiver is displayed when facing away from the viewer. Animatable.
- `mask` (page 98) *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 94) *property*  
Specifies a radius used to draw the rounded corners of the receiver's background. Animatable.
- `borderWidth` (page 90) *property*  
Specifies the width of the receiver's border. Animatable.
- `borderColor` (page 90) *property*  
The color of the receiver's border. Animatable.
- `backgroundColor` (page 89) *property*  
Specifies the background color of the receiver. Animatable.
- `backgroundFilters` (page 89) *property*  
An optional array of CoreImage filters that are applied to the receiver's background. Animatable.
- `shadowOpacity` (page 102) *property*  
Specifies the opacity of the receiver's shadow. Animatable.

[shadowRadius](#) (page 103) *property*

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

[shadowOffset](#) (page 102) *property*

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

[shadowColor](#) (page 102) *property*

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

[filters](#) (page 96) *property*

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

[compositingFilter](#) (page 91) *property*

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

[style](#) (page 103) *property*

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

[minificationFilter](#) (page 99) *property*

The filter used when reducing the size of the content.

[minificationFilterBias](#) (page 99) *property*

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

[magnificationFilter](#) (page 98) *property*

The filter used when increasing the size of the content.

## Managing the Layer Hierarchy

[sublayers](#) (page 104) *property*

An array containing the receiver's sublayers.

[superlayer](#) (page 105) *property*

Specifies receiver's superlayer. (read-only)

- [addSublayer:](#) (page 110)

Appends the layer to the receiver's [sublayers](#) (page 104) array.

- [removeFromSuperlayer](#) (page 122)

Removes the layer from the [sublayers](#) (page 104) array or [mask](#) (page 98) property of the receiver's [superlayer](#) (page 105).

- [insertSublayer:atIndex:](#) (page 118)

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

- [insertSublayer:below:](#) (page 118)

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

- [insertSublayer:above:](#) (page 118)

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

- [replaceSublayer:with:](#) (page 123)

Replaces the layer in the receiver's sublayers array with the specified new layer.

## Updating Layer Display

- [setNeedsDisplay](#) (page 125)

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

- [needsDisplayOnBoundsChange](#) (page 100) *property*  
Returns whether the receiver must be redisplayed when the bounds rectangle is updated.
- [displayIfNeeded](#) (page 115)  
Displays the layer if it has been marked as needing display.
- [needsDisplay](#) (page 120)  
Returns whether the layer has been marked as requiring display.
- + [needsDisplayForKey:](#) (page 107)  
Returns whether changes to the specified key requires the layer to be redisplayed.
- [setNeedsDisplayInRect:](#) (page 125)  
Marks the region of the receiver within the specified rectangle as needing display.

## Layer Animations

- [addAnimation:forKey:](#) (page 109)  
Add an animation object to the receiver's render tree for the specified key.
- [animationForKey:](#) (page 111)  
Returns the animation added to the receiver with the specified identifier.
- [removeAllAnimations](#) (page 121)  
Remove all animations attached to the receiver.
- [removeAnimationForKey:](#) (page 122)  
Remove the animation attached to the receiver with the specified key.
- [animationKeys](#) (page 111)  
Returns an array containing the keys of all animations currently attached to the receiver.

## Managing Layer Resizing and Layout

- [LayoutManager](#) (page 98) *property*  
Specifies the layout manager responsible for laying out the receiver's sublayers.
- [setNeedsLayout](#) (page 126)  
Called when the preferred size of the receiver may have changed.
- [constraints](#) (page 92) *property*  
Specifies the constraints used to layout the receiver's sublayers when using an `CAConstraintManager` instance as the layout manager.
- [addConstraint:](#) (page 110)  
Adds the constraint to the receiver's array of constraint objects.
- [name](#) (page 100) *property*  
The name of the receiver.
- [autoresizingMask](#) (page 89) *property*  
A bitmask defining how the layer is resized when the bounds of its superlayer changes.
- [resizeWithOldSuperlayerSize:](#) (page 124)  
Informs the receiver that the bounds size of its superview has changed.
- [resizeSublayersWithOldSize:](#) (page 123)  
Informs the receiver's sublayers that the receiver's bounds rectangle size has changed.

- [preferredFrameSize](#) (page 121)  
Returns the preferred frame size of the layer in the coordinate space of the superlayer.
- [layoutIfNeeded](#) (page 119)  
Recalculate the receiver's layout, if required.
- [layoutSublayers](#) (page 119)  
Called when the layer requires layout.
- [needsLayout](#) (page 120)  
Returns whether the layer has been marked as requiring layout.

## Actions

- [actions](#) (page 87) *property*  
A dictionary mapping keys to objects that implement the `CAAction` protocol.
- + [defaultActionForKey:](#) (page 106)  
Returns an object that implements the default action for the specified identifier.
- [actionForKey:](#) (page 108)  
Returns an object that implements the action for the specified identifier.
- [actionForLayer:forKey:](#) (page 127) *delegate method*  
Allows the delegate to customize the action for a layer.

## Mapping Between Coordinate and Time Spaces

- [convertPoint:fromLayer:](#) (page 112)  
Converts the point from the specified layer's coordinate system to the receiver's coordinate system.
- [convertPoint:toLayer:](#) (page 113)  
Converts the point from the receiver's coordinate system to the specified layer's coordinate system.
- [convertRect:fromLayer:](#) (page 113)  
Converts the rectangle from the specified layer's coordinate system to the receiver's coordinate system.
- [convertRect:toLayer:](#) (page 114)  
Converts the rectangle from the receiver's coordinate system to the specified layer's coordinate system.
- [convertTime:fromLayer:](#) (page 114)  
Converts the time interval from the specified layer's time space to the receiver's time space.
- [convertTime:toLayer:](#) (page 114)  
Converts the time interval from the receiver's time space to the specified layer's time space

## Hit Testing

- [hitTest:](#) (page 116)  
Returns the farthest descendant of the receiver in the layer hierarchy (including itself) that contains a specified point.
- [containsPoint:](#) (page 111)  
Returns whether the receiver contains a specified point.

## Rendering

- [renderInContext:](#) (page 122)  
Renders the receiver and its sublayers into the specified context.

## Scrolling

- [visibleRect](#) (page 105) *property*  
Returns the visible region of the receiver, in its own coordinate space. (read-only)
- [scrollPoint:](#) (page 124)  
Scrolls the receiver's closest ancestor `CAScrollLayer` so that the specified point lies at the origin of the layer.
- [scrollRectToVisible:](#) (page 124)  
Scrolls the receiver's closest ancestor `CAScrollLayer` the minimum distance needed so that the specified rectangle becomes visible.

## Modifying the Delegate

- [delegate](#) (page 95) *property*  
Specifies the receiver's delegate object.

## Key-Value Coding Extensions

- [shouldArchiveValueForKey:](#) (page 126)  
Specifies whether the value of the property for a given key is archived.
- + [defaultValueForKey:](#) (page 107)  
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 108) for a description of the action search pattern.

### Availability

Available in Mac OS X v10.5 and later.

**See Also**

- [actionForKey:](#) (page 108)
- [actionForLayer:forKey:](#) (page 127)
- + [defaultActionForKey:](#) (page 106)
- [@property style](#) (page 103)

**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](#)” (page 249) in *Core Animation Programming Guide* for more information on the relationship between the [bounds](#) (page 91), [anchorPoint](#) (page 88) and [position](#) (page 101) properties.

**Availability**

Available in Mac OS X v10.5 and later.

**See Also**

[@property position](#) (page 101)

**Related Sample Code**

CoreAnimationText

GeekGameBoard

LightTable

**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 Mac OS X v10.6 and later.

**See Also**

[@property anchorPoint](#) (page 88)



**Declared In**

CALayer.h

## autoresizingMask

A bitmask defining how the layer is resized when the bounds of its superlayer changes.

```
@property unsigned int autoresizingMask
```

**Discussion**

See “[Autoresizing Mask](#)” (page 128) for possible values. Default value is `kCALayerNotSizable` (page 128).

**Availability**

Available in Mac OS X v10.5 and later.

**Related Sample Code**

GeekGameBoard

LightTable

**Declared In**

CALayer.h

## backgroundColor

Specifies the background color of the receiver. Animatable.

```
@property CGColorRef backgroundColor
```

**Discussion**

The default is `nil`.

**Availability**

Available in Mac OS X v10.5 and later.

**Related Sample Code**

CoreAnimationKioskStyleMenu

Fire

Fireworks

GeekGameBoard

LightTable

**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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Related Sample Code**

GeekGameBoard

**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 92) and [sublayers](#) (page 104) and includes the effects of the [cornerRadius](#) (page 94) property. The default is 0.0.

**Availability**

Available in Mac OS X v10.5 and later.

**Related Sample Code**

GeekGameBoard

LightTable

**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](#)” (page 249) in *Core Animation Programming Guide* for more information on the relationship between the [bounds](#) (page 91), [anchorPoint](#) (page 88) and [position](#) (page 101) properties.

**Availability**

Available in Mac OS X v10.5 and later.

**Related Sample Code**

CALayerEssentials

GeekGameBoard

ImageKitDemo

LightTable

NineSlice

**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 Mac OS X v10.5 and later.

### See Also

[@property backgroundFilters](#) (page 89)

### Declared In

`CALayer.h`

## constraints

Specifies the constraints used to layout the receiver's sublayers when using an `CAConstraintManager` instance as the layout manager.

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

### Discussion

See [CAConstraintLayoutManager Class Reference](#) (page 39) for more information.

### Availability

Available in Mac OS X v10.5 and later.

### Declared In

`CAConstraintLayoutManager.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 Mac OS X v10.5 and later.

**See Also**

[@property contentsRect](#) (page 94)

**Related Sample Code**

AnimatedTableView

CoreAnimationKioskStyleMenu

GeekGameBoard

LightTable

NineSlice

**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 93) 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 Mac OS X v10.6 and later.

**See Also**

[@property contentsRect](#) (page 94)

[@property contentsGravity](#) (page 93)

[@property contents](#) (page 92)

**Related Sample Code**

NineSlice

**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 130). The default value is `kCAGravityResize` (page 132).

**Availability**

Available in Mac OS X v10.5 and later.

**Related Sample Code**

CoreAnimationKioskStyleMenu

GeekGameBoard

**Declared In**

CALayer.h

## contentsRect

A rectangle, in the unit coordinate space, defining the subrectangle of [contents](#) (page 92) 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 Mac OS X v10.5 and later.

**See Also**

[@property contents](#) (page 92)

**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 Mac OS X v10.5 and later.

**Related Sample Code**

GeekGameBoard

LightTable

**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 Mac OS X v10.5 and later.

**Related Sample Code**

CALayerEssentials

LightTable

**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 Mac OS X v10.5 and later.

**Related Sample Code**

GeekGameBoard

**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 130).

#### Availability

Available in Mac OS X v10.5 and later.

#### Related Sample Code

GeekGameBoard

#### 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 Mac OS X v10.5 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 91), [anchorPoint](#) (page 88) and [position](#) (page 101) properties. When the `frame` is set, the receiver’s [position](#) (page 101) and the size of the receiver’s [bounds](#) (page 91) are changed to match the new frame rectangle. The value of this property is specified in points.

See [“Layer Geometry and Transforms”](#) (page 249) in *Core Animation Programming Guide* for more information on the relationship between the [bounds](#) (page 91), [anchorPoint](#) (page 88) and [position](#) (page 101) properties.



**Note:** The `frame` property is not directly animatable. Instead you should animate the appropriate combination of the `bounds` (page 91), `anchorPoint` (page 88) and `position` (page 101) properties to achieve the desired result.

**Availability**

Available in Mac OS X v10.5 and later.

**Related Sample Code**

GeekGameBoard

ImageBrowserViewAppearance

ImageKitDemo

LightTable

NineSlice

**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 Mac OS X v10.6 and later.

**Related Sample Code**

CoreAnimationText

**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 Mac OS X v10.5 and later.

**Declared In**

CALayer.h

## layoutManager

Specifies the layout manager responsible for laying out the receiver's sublayers.

```
@property(retain) id layoutManager
```

**Discussion**

The `layoutManager` must implement the `CALayoutManager Informal Protocol` informal protocol. The default value is `nil`.

**Availability**

Available in Mac OS X v10.5 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 132). The default value is `kCAFilterLinear` (page 133).

**Availability**

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 94) property. If YES and a [mask](#) (page 98) property is specified, the two masks are multiplied to get the actual mask values. Defaults to NO.

**Availability**

Available in Mac OS X v10.5 and later.

**Related Sample Code**

GeekGameBoard

LightTable

**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 132). The default value is `kCAFilterLinear` (page 133).

**Availability**

Available in Mac OS X v10.5 and later.

**See Also**

[@property minificationFilterBias](#) (page 99)

**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 99) when it is set to `kCAFilterTrilinear` (page 133).

Defaults to 0.

**Availability**

Available in Mac OS X v10.6 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 Mac OS X v10.5 and later.

**Related Sample Code**

CoreAnimationText

**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 125) is automatically invoked when the receiver's [bounds](#) (page 91) is changed. Default value is NO.

**Availability**

Available in Mac OS X v10.5 and later.

**Related Sample Code**

CALayerEssentials

**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 Mac OS X v10.5 and later.

**Related Sample Code**

GeekGameBoard

**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 Mac OS X v10.5 and later.

**Related Sample Code**

LightTable

**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 88). The value of this property is specified in points. The default is (0.0, 0.0).

See “[Layer Geometry and Transforms](#)” (page 249) in *Core Animation Programming Guide* for more information on the relationship between the [bounds](#) (page 91), [anchorPoint](#) (page 88) and [position](#) (page 101) properties.

**Availability**

Available in Mac OS X v10.5 and later.

**See Also**

[@property anchorPoint](#) (page 88)

**Related Sample Code**

CALayerEssentials

CoreAnimationKioskStyleMenu

CoreAnimationText

GeekGameBoard

LightTable

**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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Related Sample Code**

AnimatedTableView

**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 Mac OS X v10.5 and later.

**Related Sample Code**

AnimatedTableView

**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 107) 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 Mac OS X v10.5 and later.

**Related Sample Code**

CoreAnimationText

**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 105) set to `nil`.

**Availability**

Available in Mac OS X v10.5 and later.

**Related Sample Code**

GeekGameBoard

**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 Mac OS X v10.5 and later.

**Related Sample Code**

CALayerEssentials

**Declared In**

CALayer.h



## superlayer

Specifies receiver's superlayer. (read-only)

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

### Availability

Available in Mac OS X v10.5 and later.

### Related Sample Code

GeekGameBoard

LightTable

### 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 Mac OS X v10.5 and later.

### Related Sample Code

AnimatedTableView

CoreAnimationText

GeekGameBoard

### 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 Mac OS X v10.5 and later.

### Related Sample Code

ImageBrowserViewAppearance

ImageKitDemo

**Declared In**

CASScrollLayer.h

**zPosition**

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

```
@property CGFloat zPosition
```

**Discussion**

Defaults to 0.

**Availability**

Available in Mac OS X v10.5 and later.

**Related Sample Code**

GeekGameBoard

**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 108) for a description of the action search pattern.

**Availability**

Available in Mac OS X v10.5 and later.

**See Also**

- [actionForKey:](#) (page 108)
- [actionForLayer:forKey:](#) (page 127)
- [@property actions](#) (page 87)
- [@property style](#) (page 103)

**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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

### Related Sample Code

`CALayerEssentials`

`GeekGameBoard`

`ImageBrowserViewAppearance`

`ImageKitDemo`

`LightTable`

### 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 Mac OS X v10.6 and later.

**See Also**

+ [defaultActionForKey:](#) (page 106)

+ [defaultValueForKey:](#) (page 107)

**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 127).

- Return the object that corresponds to the identifier in the receiver's `actions` (page 87) dictionary property.
- If `nil` is returned there is no action specified for requested `aKey`.
- Search the `style` (page 103) dictionary recursively for an actions dictionary that contains the identifier.
- Call the receiver's `defaultActionForKey:` (page 106) 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 Mac OS X v10.5 and later.

#### See Also

- `actionForLayer:forKey:` (page 127)
- [@property actions](#) (page 87)
- + `defaultActionForKey:` (page 106)
- [@property style](#) (page 103)

#### 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 130) 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 `kCATransitionAnimationDuration` (page 192) transaction property, otherwise .25 seconds

#### Availability

Available in Mac OS X v10.5 and later.

#### Related Sample Code

`CoreAnimationKioskStyleMenu`

#### Declared In

`CALayer.h`

## addConstraint:

Adds the constraint to the receiver's array of constraint objects.

```
- (void)addConstraint:(CAConstraint *)aConstraint
```

### Parameters

*aConstraint*

The constraint object to add to the receiver's array of constraint objects.

### Discussion

See [CAConstraintLayoutManager Class Reference](#) (page 39) for more information.

### Availability

Available in Mac OS X v10.5 and later.

### Related Sample Code

CoreAnimationKioskStyleMenu

### Declared In

CAConstraintLayoutManager.h

## addSublayer:

Appends the layer to the receiver's [sublayers](#) (page 104) array.

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

### Parameters

*aLayer*

The layer to be added to the receiver's [sublayers](#) (page 104) array.

### Availability

Available in Mac OS X v10.5 and later.

### Related Sample Code

CALayerEssentials

CoreAnimationKioskStyleMenu

GeekGameBoard

ImageBrowserViewAppearance

ImageKitDemo

### Declared In

CALayer.h

## affineTransform

Convenience method for getting the [transform](#) (page 105) property as an affine transform.

```
- (CGAffineTransform)affineTransform
```

### Return Value

A `CGAffineTransform` instance that best represents the receiver's [transform](#) (page 105) property.

**Availability**

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.6 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 Mac OS X v10.5 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 116) by the default `display` (page 115) method will have been y- flipped and rectangles passed to `setNeedsDisplayInRect:` (page 125) will be similarly flipped.

Defaults to NO.

Subclasses should not attempt to redefine this method.

**Availability**

Available in Mac OS X v10.6 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 Mac OS X v10.5 and later.

**Related Sample Code**

GeekGameBoard

**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 Mac OS X v10.5 and later.

**Related Sample Code**

LightTable

**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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Declared In**

CALayer.h

## display

Reload the content of this layer.

```
- (void)display
```

**Discussion**

Calls the [drawInContext:](#) (page 116) method, then updates the receiver's [contents](#) (page 92) property. You should not call this method directly.

Subclasses can override this method to set the [contents](#) (page 92) property to an appropriate `CGImageRef`.

**Availability**

Available in Mac OS X v10.5 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 115) if it has been marked as requiring display.

**Availability**

Available in Mac OS X v10.6 and later.

**See Also**

- [needsDisplay](#) (page 120)

**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 115) method when the `contents` (page 92) 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

**See Also**

+ [Layer](#) (page 107)

**Related Sample Code**

CoreAnimationText

GeekGameBoard

ImageBrowserViewAppearance

ImageKitDemo

LightTable

**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 Mac OS X v10.5 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 104) array, an exception is raised.

**Availability**

Available in Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Related Sample Code**

Core Animation QuickTime Layer

GeekGameBoard

**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 104) array, an exception is raised.

**Availability**

Available in Mac OS X v10.5 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 an ancestor layer is found that does not require layout. Then layout is performed on the entire layer-tree beneath that ancestor.

**Availability**

Available in Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Related Sample Code**

CoreAnimationText

**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 Mac OS X v10.5 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 Mac OS X v10.6 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 Mac OS X v10.6 and later.

### See Also

- [setNeedsLayout](#) (page 126)

### 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 91) rectangle mapped into coordinate space of the receiver's [superlayer](#) (page 105).

### Availability

Available in Mac OS X v10.5 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 104), [mask](#) (page 98), and [superlayer](#) (page 105) 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 116) message to the `presentationLayer` will query the presentation values of the layer tree.

### Availability

Available in Mac OS X v10.5 and later.

### Declared In

`CALayer.h`

## removeAllAnimations

Remove all animations attached to the receiver.

```
- (void)removeAllAnimations
```

### Availability

Available in Mac OS X v10.5 and later.

### Related Sample Code

`AnimatedTableView`

**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 Mac OS X v10.5 and later.

**Declared In**

CALayer.h

**removeFromSuperlayer**

Removes the layer from the [sublayers](#) (page 104) array or [mask](#) (page 98) property of the receiver's [superlayer](#) (page 105).

```
- (void)removeFromSuperlayer
```

**Availability**

Available in Mac OS X v10.5 and later.

**Related Sample Code**

GeekGameBoard

LightTable

**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 89), [filters](#) (page 96), [compositingFilter](#) (page 91), or a [mask](#) (page 98) values. Future versions of Mac OS X may add support for rendering these layers and properties.

**Availability**

Available in Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Declared In**

`CALayer.h`

**resizeSublayersWithOldSize:**

Informs the receiver's sublayers that the receiver's bounds rectangle size has changed.

```
- (void)resizeSublayersWithOldSize:(CGSize)size
```

**Parameters**

*size*

The previous size of the receiver's bounds rectangle.

**Discussion**

This method is used when the `autoresizingmask` property is used for resizing. It is called when the receiver's `bounds` property is altered. It calls `resizeSublayersWithOldSize:` on each sublayer to resize the sublayer's frame to match the new superlayer bounds based on the sublayer's autoresizing mask.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CALayer.h

**resizeWithOldSuperlayerSize:**

Informs the receiver that the bounds size of its superview has changed.

```
- (void)resizeWithOldSuperlayerSize:(CGSize)size
```

**Parameters***size*

The previous size of the superlayer's bounds rectangle

**Discussion**

This method is used when the `autoresizingmask` property is used for resizing. It is called when the receiver's `bounds` property is altered. It calls `resizeWithOldSuperlayerSize:` on each sublayer to resize the sublayer's frame to match the new superlayer bounds based on the sublayer's autoresizing mask.

**Availability**

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Declared In**

CAScrollLayer.h

**setAffineTransform:**

Convenience method for setting the [transform](#) (page 105) property as an affine transform.

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

**Parameters**

*m*

The affine transform to set as the [transform](#) (page 105) property.

**Availability**

Available in Mac OS X v10.5 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 116) which may result in the delegate receiving either a [displayLayer:](#) (page 127) or [drawLayer:inContext:](#) (page 128) message.

**Availability**

Available in Mac OS X v10.5 and later.

**Related Sample Code**

CALayerEssentials

GeekGameBoard

**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 Mac OS X v10.5 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 119) on the receiver and all its superlayers). If the receiver's layout manager implements the `invalidateLayoutOfLayer:` method it is called.

**Availability**

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 108) for a description of the action search pattern.

#### **Availability**

Available in Mac OS X v10.5 and later.

#### **See Also**

- [actionForLayer:forKey:](#) (page 127)
- [@property actions](#) (page 87)
- + [defaultActionForKey:](#) (page 106)
- [@property style](#) (page 103)

#### **Declared In**

`CALayer.h`

### **displayLayer:**

Allows the delegate to override the `display` (page 115) 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 Mac OS X v10.5 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 116).

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

`CALayer.h`

## Constants

**Autoresizing Mask**

These constants are used by the `autoresizingMask` (page 89) property.

```
enum CAAutoresizingMask
{
    kCALayerNotSizable      = 0,
    kCALayerMinXMargin     = 1U << 0,
    kCALayerWidthSizable   = 1U << 1,
    kCALayerMaxXMargin     = 1U << 2,
    kCALayerMinYMargin     = 1U << 3,
    kCALayerHeightSizable  = 1U << 4,
    kCALayerMaxYMargin     = 1U << 5
};
```

**Constants**

`kCALayerNotSizable`

The receiver cannot be resized.

Available in Mac OS X v10.5 and later.

Declared in `CALayer.h`.

`kCALayerMinXMargin`

The left margin between the receiver and its superview is flexible.

Available in Mac OS X v10.5 and later.

Declared in `CALayer.h`.



`kCALayerWidthSizable`

The receiver's width is flexible.

Available in Mac OS X v10.5 and later.

Declared in `CALayer.h`.

`kCALayerMaxXMargin`

The right margin between the receiver and its superview is flexible.

Available in Mac OS X v10.5 and later.

Declared in `CALayer.h`.

`kCALayerMinYMargin`

The bottom margin between the receiver and its superview is flexible.

Available in Mac OS X v10.5 and later.

Declared in `CALayer.h`.

`kCALayerHeightSizable`

The receiver's height is flexible.

Available in Mac OS X v10.5 and later.

Declared in `CALayer.h`.

`kCALayerMaxYMargin`

The top margin between the receiver and its superview is flexible.

Available in Mac OS X v10.5 and later.

Declared in `CALayer.h`.

#### Declared In

`CALayer.h`

## Action Identifiers

These constants are the predefined action identifiers used by [`actionForKey:`](#) (page 108), [`addAnimation:forKey:`](#) (page 109), [`defaultActionForKey:`](#) (page 106), [`removeAnimationForKey:`](#) (page 122), [`actionForLayer:forKey:`](#) (page 127), and the `CAAction` protocol method [`runActionForKey:object:arguments:`](#) (page 217).

```
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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CALayer.h`.

`kCATransition`

The identifier that represents a transition animation.

Available in Mac OS X v10.5 and later.

Declared in `CALayer.h`.

**Declared In**

`CALayer.h`

## Edge Antialiasing Mask

This mask is used by the `edgeAntialiasingMask` (page 95) 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 Mac OS X v10.5 and later.

Declared in `CALayer.h`.

`kCALayerRightEdge`

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

Available in Mac OS X v10.5 and later.

Declared in `CALayer.h`.

`kCALayerBottomEdge`

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

Available in Mac OS X v10.5 and later.

Declared in `CALayer.h`.

`kCALayerTopEdge`

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

Available in Mac OS X v10.5 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 93) 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 Mac OS X v10.5 and later.

Declared in `CALayer.h`.

`kCAGravityTop`

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

Available in Mac OS X v10.5 and later.

Declared in `CALayer.h`.

`kCAGravityBottom`

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

Available in Mac OS X v10.5 and later.

Declared in `CALayer.h`.

`kCAGravityLeft`

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

Available in Mac OS X v10.5 and later.

Declared in `CALayer.h`.

`kCAGravityRight`

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

Available in Mac OS X v10.5 and later.

Declared in `CALayer.h`.

`kCAGravityTopLeft`

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

Available in Mac OS X v10.5 and later.

Declared in `CALayer.h`.

`kCAGravityTopRight`

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

Available in Mac OS X v10.5 and later.

Declared in `CALayer.h`.

`kCAGravityBottomLeft`

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

Available in Mac OS X v10.5 and later.

Declared in `CALayer.h`.

`kCAGravityBottomRight`

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

Available in Mac OS X v10.5 and later.

Declared in `CALayer.h`.

`kCAGravityResize`

The content is resized to fit the entire bounds rectangle.

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in `CATransform3D.h`.

#### Declared In

`CATransform3D.h`

## Scaling Filters

These constants specify the scaling filters used by [magnificationFilter](#) (page 98) and [minificationFilter](#) (page 99).

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

**Constants**

`kCAFilterLinear`

Linear interpolation filter.

Available in Mac OS X v10.5 and later.

Declared in `CALayer.h`.

`kCAFilterNearest`

Nearest neighbor interpolation filter.

Available in Mac OS X v10.5 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 Mac OS X v10.6 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 Mac OS X v10.5 and later.

**Declared In**

`CATransform3D.h`

# CAMediaTimingFunction Class Reference

---

<b>Inherits from</b>	NSObject
<b>Conforms to</b>	NSCoding NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.5 and later.
<b>Declared in</b>	CAMediaTimingFunction.h
<b>Companion guides</b>	Core Animation Programming Guide Core Animation Cookbook
<b>Related sample code</b>	CocoaSlides CoreAnimationKioskStyleMenu

## 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 136)

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

+ `functionWithControlPoints:::::` (page 136)

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

- `initWithControlPoints:::::` (page 137)

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

## Accessing the Control Points

- [getControlPointAtIndex:values:](#) (page 137)  
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 Mac OS X v10.5 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 138).

**Return Value**

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

**Availability**

Available in Mac OS X v10.5 and later.

**Related Sample Code**

CocoaSlides

CoreAnimationKioskStyleMenu

**Declared In**

`CAMediaTimingFunction.h`

## Instance Methods

### **getControlPointAtIndex:values:**

Returns the control point for the specified index.

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

**Parameters***index*

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

*ptr*

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

**Discussion**

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

**Availability**

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 136).

```
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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.6 and later.

Declared in `CAMediaTimingFunction.h`.



# CAOpenGLLayer Class Reference

---

<b>Inherits from</b>	CALayer : NSObject
<b>Conforms to</b>	NSCoding (CALayer) CAMediaTiming (CALayer) NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.5 and later.
<b>Declared in</b>	CAOpenGLLayer.h
<b>Companion guides</b>	Core Animation Programming Guide Core Animation Cookbook
<b>Related sample code</b>	CALayerEssentials

## Overview

CAOpenGLLayer provides a layer suitable for rendering OpenGL content.

To provide OpenGL content you subclass CAOpenGLLayer and override [drawInCGLContext:pixelFormat:forLayerTime:displayTime:](#) (page 144). You can specify that the OpenGL content is static by setting the [asynchronous](#) (page 142) property to NO.

## Tasks

### Drawing the Content

- [asynchronous](#) (page 142) *property*  
Determines when the contents of the layer are updated.
- [isAsynchronous](#) (page 145)  
A synthesized accessor for the [asynchronous](#) (page 142) property.
- [canDrawInCGLContext:pixelFormat:forLayerTime:displayTime:](#) (page 143)  
Returns whether the receiver should draw OpenGL content for the specified time.
- [drawInCGLContext:pixelFormat:forLayerTime:displayTime:](#) (page 144)  
Draws the OpenGL content for the specified time.

## Managing the Pixel Format

- [copyCGLPixelFormatForDisplayMask:](#) (page 144)  
Returns the OpenGL pixel format suitable for rendering to the set of displays specified by the display mask.
- [releaseCGLPixelFormat:](#) (page 145)  
Releases the specified OpenGL pixel format object.

## Managing the Rendering Context

- [copyCGLContextForPixelFormat:](#) (page 143)  
Returns the rendering context the receiver requires for the specified pixel format.
- [releaseCGLContext:](#) (page 145)  
Releases the specified rendering context.

## Properties

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

### asynchronous

Determines when the contents of the layer are updated.

@property BOOL asynchronous

#### Discussion

If NO, the contents of the layer are updated only in response to receiving a [setNeedsDisplay](#) (page 125) message. When YES, the receiver’s [canDrawInCGLContext:pixelFormat:forLayerTime:displayTime:](#) (page 143) is called periodically to determine if the OpenGL content should be updated.

#### Availability

Available in Mac OS X v10.5 and later.

#### See Also

- [isAsynchronous](#) (page 145)

#### Related Sample Code

CALayerEssentials

#### Declared In

CAOpenGLLayer.h

## Instance Methods

### canDrawInCGLContext:pixelFormat:forLayerTime:displayTime:

Returns whether the receiver should draw OpenGL content for the specified time.

```
- (BOOL)canDrawInCGLContext:(CGLContextObj)glContext
  pixelFormat:(CGLPixelFormatObj)pixelFormat
  forLayerTime:(CTimeInterval)timeInterval
  displayTime:(const CVTimeStamp *)timeStamp
```

#### Parameters

*glContext*

The `CGLContextObj` in to which the OpenGL content would be drawn.

*pixelFormat*

The pixel format used when the *glContext* was created.

*timeInterval*

The current layer time.

*timeStamp*

The display timestamp associated with *timeInterval*. Can be null.

#### Return Value

YES if the receiver should render OpenGL content, NO otherwise.

#### Discussion

This method is called before attempting to render the frame for the layer time specified by *timeInterval*. If the method returns NO, the frame is skipped. The default implementation always returns YES.

#### Availability

Available in Mac OS X v10.5 and later.

#### Declared In

CAOpenGLLayer.h

### copyCGLContextForPixelFormat:

Returns the rendering context the receiver requires for the specified pixel format.

```
- (CGLContextObj)copyCGLContextForPixelFormat:(CGLPixelFormatObj)pixelFormat
```

#### Parameters

*pixelFormat*

The pixel format for the rendering context.

#### Return Value

A new `CGLContext` with renderers for *pixelFormat*.

#### Discussion

This method is called when a rendering context is needed by the receiver. The default implementation allocates a new context with a null share context.

You should not call this method directly, it is intended to be overridden by subclasses.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CAOpenGLLayer.h

**copyCGLPixelFormatForDisplayMask:**

Returns the OpenGL pixel format suitable for rendering to the set of displays specified by the display mask.

```
- (CGLPixelFormatObj)copyCGLPixelFormatForDisplayMask:(uint32_t)mask
```

**Parameters**

*mask*

The display mask the OpenGL content will be rendered on.

**Discussion**

This method is called when a pixel format object is needed for the receiver. The default implementation returns a 32bpp fixed point pixel format, with the `NoRecovery` and `Accelerated` flags set.

You should not call this method directly, it is intended to be overridden by subclasses.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CAOpenGLLayer.h

**drawInCGLContext:pixelFormat:forLayerTime:displayTime:**

Draws the OpenGL content for the specified time.

```
- (void)drawInCGLContext:(CGLContextObj)glContext
    pixelFormat:(CGLPixelFormatObj)pixelFormat
    forLayerTime:(CFTimeInterval)timeInterval
    displayTime:(const CVTimeStamp *)timeStamp
```

**Parameters**

*glContext*

The rendering context in to which the OpenGL content should be rendered.

*pixelFormat*

The pixel format used when the *glContext* was created.

*timeInterval*

The current layer time.

*timeStamp*

The display timestamp associated with *timeInterval*. Can be null.

**Discussion**

This method is called when a new frame needs to be generated for the layer time specified by *timeInterval*. The viewport of *glContext* is set correctly for the size of the layer. No other state is defined. If the method enables OpenGL features, it should disable them before returning.



The default implementation of the method flushes the context.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CAOpenGLLayer.h

## isAsynchronous

A synthesized accessor for the [asynchronous](#) (page 142) property.

- (BOOL)isAsynchronous

**See Also**

[@property asynchronous](#) (page 142)

## releaseCGLContext:

Releases the specified rendering context.

- (void)releaseCGLContext:(CGLContextObj)*glContext*

**Parameters**

*glContext*

The rendering context to release.

**Discussion**

This method is called when the OpenGL context that was previously returned by [copyCGLContextForPixelFormat:](#) (page 143) is no longer needed.

You should not call this method directly, it is intended to be overridden by subclasses.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CAOpenGLLayer.h

## releaseCGLPixelFormat:

Releases the specified OpenGL pixel format object.

- (void)releaseCGLPixelFormat:(CGLPixelFormatObj)*pixelFormat*

**Parameters**

*pixelFormat*

The pixel format object to release.

**Discussion**

This method is called when the OpenGL pixel format that was previously returned by [copyCGLContextForPixelFormat:](#) (page 143).

You should not call this method directly, it is intended to be overridden by subclasses.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CAOpenGLLayer.h

# CAPROPERTYANIMATION Class Reference

---

<b>Inherits from</b>	CAAnimation : NSObject
<b>Conforms to</b>	NSCoding (CAAnimation) NSCopying (CAAnimation) CAAction (CAAnimation) CAMediaTiming (CAAnimation) NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.5 and later.
<b>Declared in</b>	CAAnimation.h
<b>Companion guides</b>	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 149) *property*

Specifies the key path the receiver animates.

### Property Value Calculation Behavior

[cumulative](#) (page 148) *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 148) *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 149) *property*

An optional value function that is applied to interpolated values.

## Creating an Animation

+ `animationWithKeyPath:` (page 149)

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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Related Sample Code**

CoreAnimationKioskStyleMenu

**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 Mac OS X v10.6 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 Mac OS X v10.5 and later.

**Declared In**

`CAAnimation.h`

# CARenderer Class Reference

---

<b>Inherits from</b>	NSObject
<b>Conforms to</b>	NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.5 and later.
<b>Declared in</b>	CARenderer.h
<b>Companion guides</b>	Core Animation Programming Guide Core Animation Cookbook

## Overview

`CARenderer` allows an application to render a layer tree into a CGL context. For real-time output you should use an instance of `NSView` to host the layer-tree.

## Tasks

### Rendered Layer

[layer](#) (page 152) *property*

The root layer of the layer-tree the receiver should render.

### Renderer Geometry

[bounds](#) (page 152) *property*

The bounds of the receiver.

### Create a New Renderer

+ [rendererWithCGLContext:options:](#) (page 153)

Creates and returns a `CARenderer` instance with the render target specified by the Core OpenGL context.

## Render a Frame

- [beginFrameAtTime:timeStamp:](#) (page 153)  
Begin rendering a frame at the specified time.
- [updateBounds](#) (page 155)  
Returns the bounds of the update region that contains all pixels that will be rendered by the current frame.
- [addUpdateRect:](#) (page 153)  
Adds the rectangle to the update region of the current frame.
- [render](#) (page 154)  
Render the update region of the current frame to the target context.
- [nextFrameTime](#) (page 154)  
Returns the time at which the next update should happen.
- [endFrame](#) (page 154)  
Release any data associated with the current frame.

## Properties

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

### bounds

The bounds of the receiver.

```
@property CGRect bounds
```

#### Availability

Available in Mac OS X v10.5 and later.

#### Declared In

CARenderer.h

### layer

The root layer of the layer-tree the receiver should render.

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

#### Availability

Available in Mac OS X v10.5 and later.

#### Declared In

CARenderer.h



## Class Methods

### **rendererWithCGLContext:options:**

Creates and returns a `CARenderer` instance with the render target specified by the Core OpenGL context.

```
+ (CARenderer *)rendererWithCGLContext:(void *)ctx
  options:(NSDictionary *)dict
```

#### Parameters

*ctx*

A Core OpenGL render context that is used as the render target.

*dict*

A dictionary of optional parameters.

#### Return Value

A new instance of `CARenderer` that will use *ctx* as the render target.

#### Availability

Available in Mac OS X v10.5 and later.

#### Declared In

`CARenderer.h`

## Instance Methods

### **addUpdateRect:**

Adds the rectangle to the update region of the current frame.

```
- (void)addUpdateRect:(CGRect)aRect
```

#### Parameters

*aRect*

A rectangle defining the region to be added to the update region.

#### Availability

Available in Mac OS X v10.5 and later.

#### Declared In

`CARenderer.h`

### **beginFrameAtTime:timeStamp:**

Begin rendering a frame at the specified time.

```
- (void)beginFrameAtTime:(CFTimeInterval)timeInterval
  timeStamp:(CVTimeStamp *)timeStamp
```

**Parameters***timeInterval*

The layer time.

*timeStamp*The display timestamp associated with *timeInterval*. Can be null.**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CARenderer.h

**endFrame**

Release any data associated with the current frame.

- (void)endFrame

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CARenderer.h

**nextFrameTime**

Returns the time at which the next update should happen.

- (CFTimeInterval)nextFrameTime

**Return Value**

The time at which the next update should happen.

**Discussion**

If infinite, no update needs to be scheduled yet. If *nextFrameTime* is the current frame time, a continuous animation is running and an update should be scheduled after an appropriate delay.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CARenderer.h

**render**

Render the update region of the current frame to the target context.

- (void)render

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CARenderer.h

**updateBounds**

Returns the bounds of the update region that contains all pixels that will be rendered by the current frame.

- (CGRect)updateBounds

**Return Value**

The bounds of the update region..

**Discussion**

Initially `updateBounds` will include all differences between the current frame and the previously rendered frame.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CARenderer.h



# CAReplicatorLayer Class Reference

---

<b>Inherits from</b>	CALayer : NSObject
<b>Conforms to</b>	NSCoding (CALayer) CAMediaTiming (CALayer) NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.6 and later.
<b>Declared in</b>	CAReplicatorLayer.h
<b>Companion guides</b>	Core Animation Programming Guide Core Animation Cookbook

## Overview

The `CAReplicatorLayer` class creates a specified number of copies of its sublayers (the source layer), each copy potentially having geometric, temporal and color transformations applied to it.

**Note:** The `CAReplicatorLayer` implementation of `hitTest:` (page 116) currently only tests the first instance of z replicator layer's sublayers. This may change in the future.

## Tasks

### Setting Instance Display Properties

`instanceCount` (page 159) *property*

The number of copies to create, including the source layers.

`instanceDelay` (page 159) *property*

Specifies the delay, in seconds, between replicated copies. Animatable.

`instanceTransform` (page 160) *property*

The transform matrix applied to the previous instance to produce the current instance. Animatable.

## Modifying Instance Layer Geometry

[preservesDepth](#) (page 161) *property*

Defines whether this layer flattens its sublayers into its plane.

## Accessing Instance Color Values

[instanceColor](#) (page 159) *property*

Defines the color used to multiply the source object. Animatable.

[instanceRedOffset](#) (page 160) *property*

Defines the offset added to the red component of the color for each replicated instance. Animatable.

[instanceGreenOffset](#) (page 160) *property*

Defines the offset added to the green component of the color for each replicated instance. Animatable.

[instanceBlueOffset](#) (page 158) *property*

Defines the offset added to the blue component of the color for each replicated instance. Animatable.

[instanceAlphaOffset](#) (page 158) *property*

Defines the offset added to the alpha component of the color for each replicated instance. Animatable.

## Properties

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

### instanceAlphaOffset

Defines the offset added to the alpha component of the color for each replicated instance. Animatable.

```
@property float instanceAlphaOffset
```

#### Discussion

The `instanceAlphaOffset` is added to the alpha color component of instance  $k-1$  to produce the modulation color of instance  $k$ .

Default is 0.0.

#### Availability

Available in Mac OS X v10.6 and later.

#### Declared In

CAReplicatorLayer.h

### instanceBlueOffset

Defines the offset added to the blue component of the color for each replicated instance. Animatable.

```
@property float instanceBlueOffset
```

**Discussion**

The `instanceBlueOffset` is added to the blue color component of instance  $k-1$  to produce the modulation color of instance  $k$ .

Default is 0.0.

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAReplicatorLayer.h

## instanceColor

Defines the color used to multiply the source object. Animatable.

```
@property CGColorRef instanceColor
```

**Discussion**

Defaults to opaque white.

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAReplicatorLayer.h

## instanceCount

The number of copies to create, including the source layers.

```
@property NSInteger instanceCount
```

**Discussion**

Default value is 1, no extra copies are created.

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAReplicatorLayer.h

## instanceDelay

Specifies the delay, in seconds, between replicated copies. Animatable.

```
@property CTimeInterval instanceDelay
```

**Discussion**

Default is 0.0.

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAReplicatorLayer.h

## instanceGreenOffset

Defines the offset added to the green component of the color for each replicated instance. Animatable.

```
@property float instanceGreenOffset
```

**Discussion**

The `instanceGreenOffset` is added to the green color component of instance  $k-1$  to produce the modulation color of instance  $k$ .

Default is 0.0.

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAReplicatorLayer.h

## instanceRedOffset

Defines the offset added to the red component of the color for each replicated instance. Animatable.

```
@property float instanceRedOffset
```

**Discussion**

The `instanceRedOffset` is added to the red color component of instance  $k-1$  to produce the modulation color of instance  $k$ .

Default is 0.0.

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAReplicatorLayer.h

## instanceTransform

The transform matrix applied to the previous instance to produce the current instance. Animatable.

```
@property CATransform3D instanceTransform
```

**Discussion**

This transform matrix is applied to instance  $k-1$  to produce instance  $k$ . The matrix is applied relative to the center of this layer.

Defaults to the identity matrix.



**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAReplicatorLayer.h

**preservesDepth**

Defines whether this layer flattens its sublayers into its plane.

@property BOOL preservesDepth

**Discussion**

If YES, the layer acts similarly to the CATransformLayer and has the same restrictions.

Default is NO.

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAReplicatorLayer.h



# CAScrollView Class Reference

---

<b>Inherits from</b>	CALayer : NSObject
<b>Conforms to</b>	NSCoding (CALayer) CAMediaTiming (CALayer) NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.5 and later.
<b>Declared in</b>	CAScrollView.h
<b>Companion guides</b>	Core Animation Programming Guide Core Animation Cookbook
<b>Related sample code</b>	CALayerEssentials

## Overview

The `CAScrollView` class is a subclass of `CALayer` that simplifies displaying a portion of a layer. The extent of the scrollable area of the `CAScrollView` 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. `CAScrollView` does not provide keyboard or mouse event-handling, nor does it provide visible scrollers.

## Tasks

### Scrolling Constraints

- `scrollMode` (page 164) *property*  
Defines the axes in which the layer may be scrolled.

### Scrolling the Layer

- `scrollToPoint:` (page 164)  
Changes the origin of the receiver to the specified point.
- `scrollToRect:` (page 164)  
Scroll the contents of the receiver to ensure that the rectangle is visible.

## 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 165). The default is `kCAScrollBoth`.

#### Availability

Available in Mac OS X v10.5 and later.

#### Declared In

`CAScrollView.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 Mac OS X v10.5 and later.

#### Declared In

`CAScrollView.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 Mac OS X v10.5 and later.

**Related Sample Code**  
CALayerEssentials

**Declared In**  
CAScrollLayer.h

## Constants

### Scroll Modes

These constants describe the supported scroll modes used by the [scrollMode](#) (page 164) property.

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

#### Constants

kCAScrollNone

The receiver is unable to scroll.

Available in Mac OS X v10.5 and later.

Declared in CAScrollLayer.h.

kCAScrollVertically

The receiver is able to scroll vertically.

Available in Mac OS X v10.5 and later.

Declared in CAScrollLayer.h.

kCAScrollHorizontally

The receiver is able to scroll horizontally.

Available in Mac OS X v10.5 and later.

Declared in CAScrollLayer.h.

kCAScrollBoth

The receiver is able to scroll both horizontally and vertically.

Available in Mac OS X v10.5 and later.

Declared in CAScrollLayer.h.

#### Declared In

CAScrollLayer.h



# CAShapeLayer Class Reference

---

<b>Inherits from</b>	CALayer : NSObject
<b>Conforms to</b>	NSCoding (CALayer) CAMediaTiming (CALayer) NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.6 and later.
<b>Declared in</b>	CAShapeLayer.h
<b>Related sample code</b>	CoreAnimationText

## Overview

The `CAShapeLayer` class draws a cubic Bezier spline in its coordinate space. The shape is composited between the layer's contents and its first sublayer.

The shape will be drawn antialiased, and whenever possible it will be mapped into screen space before being rasterized to preserve resolution independence. However, certain kinds of image processing operations, such as CoreImage filters, applied to the layer or its ancestors may force rasterization in a local coordinate space.

**Note:** Shape rasterization may favor speed over accuracy. For example, pixels with multiple intersecting path segments may not give exact results.

## Tasks

### Specifying the Shape Path

[path](#) (page 172) *property*

The path defining the shape to be rendered. Animatable.

### Accessing Shape Style Properties

[fillColor](#) (page 168) *property*

The color used to fill the shape's path. Animatable.

[fillRule](#) (page 168) *property*

The fill rule used when filling the shape's path.

[lineCap](#) (page 169) *property*

Specifies the line cap style for the shape's path.

[lineDashPattern](#) (page 169) *property*

The dash pattern applied to the shape's path when stroked.

[lineDashPhase](#) (page 170) *property*

The dash phase applied to the shape's path when stroked. Animatable.

[lineJoin](#) (page 170) *property*

Specifies the line join style for the shape's path.

[lineWidth](#) (page 171) *property*

Specifies the line width of the shape's path. Animatable.

[miterLimit](#) (page 171) *property*

The miter limit used when stroking the shape's path. Animatable.

[strokeColor](#) (page 172) *property*

The color used to stroke the shape's path. Animatable.

## Properties

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

### fillColor

The color used to fill the shape's path. Animatable.

```
@property CGColorRef fillColor
```

#### Discussion

Setting `fillColor` to `nil` results in no fill being rendered.

Default is opaque black.

#### Availability

Available in Mac OS X v10.6 and later.

#### Declared In

`CAShapeLayer.h`

### fillRule

The fill rule used when filling the shape's path.



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

**Discussion**

The possible values are shown in “[Shape Fill Mode Values](#)” (page 172). The default is `kCAFillRuleNonZero` (page 173). See “Winding Rules” in *Cocoa Drawing Guide* for examples of the two fill rules.

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

`CAShapeLayer.h`

## lineCap

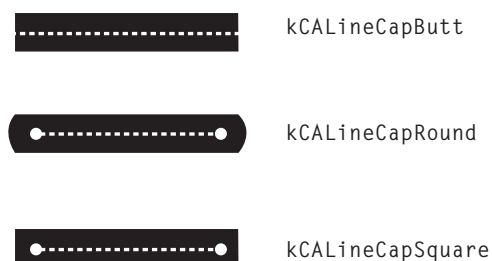
Specifies the line cap style for the shape’s path.

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

**Discussion**

The line cap style specifies the shape of the endpoints of an open path when stroked. The supported values are described in “[Line Cap Values](#)” (page 173). [Figure 17-1](#) (page 169) shows the appearance of the available line cap styles.

**Figure 17-1** Line cap styles



The default is `kCALineCapButt` (page 174).

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

`CAShapeLayer.h`

## lineDashPattern

The dash pattern applied to the shape’s path when stroked.

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

**Discussion**

The dash pattern is specified as an array of `NSNumber` objects that specify the lengths of the painted segments and unpainted segments, respectively, of the dash pattern.

For example, passing an array with the values `[2, 3]` sets a dash pattern that alternates between a 2-user-space-unit-long painted segment and a 3-user-space-unit-long unpainted segment. Passing the values `[1, 3, 4, 2]` sets the pattern to a 1-unit painted segment, a 3-unit unpainted segment, a 4-unit painted segment, and a 2-unit unpainted segment.

Default is `nil`, a solid line.

**Availability**

Available in Mac OS X v10.6 and later.

**See Also**

[@property lineDashPhase](#) (page 170)

**Declared In**

`CAShapeLayer.h`

## lineDashPhase

The dash phase applied to the shape's path when stroked. Animatable.

@property CGFloat lineDashPhase

**Discussion**

Line dash phase specifies how far into the dash pattern the line starts.

Default is 0.

**Availability**

Available in Mac OS X v10.6 and later.

**See Also**

[@property lineDashPattern](#) (page 169)

**Declared In**

`CAShapeLayer.h`

## lineJoin

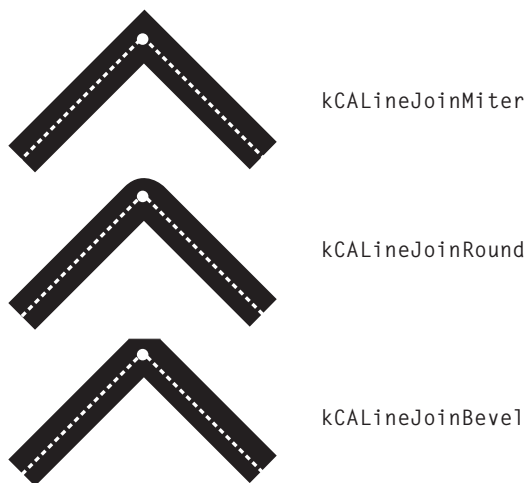
Specifies the line join style for the shape's path.

@property(copy) NSString \*lineJoin

**Discussion**

The line join style specifies the shape of the joints between connected segments of a stroked path. The supported values are described in [“Line Join Values”](#) (page 173). [Figure 17-2](#) (page 171) shows the appearance of the available line join styles.

Figure 17-2 Line join styles



The default is [kCALineJoinMiter](#) (page 173).

#### Availability

Available in Mac OS X v10.6 and later.

#### Declared In

CAShapeLayer.h

## lineWidth

Specifies the line width of the shape's path. Animatable.

@property CGFloat lineWidth

#### Availability

Available in Mac OS X v10.6 and later.

#### Declared In

CAShapeLayer.h

## miterLimit

The miter limit used when stroking the shape's path. Animatable.

@property CGFloat miterLimit

#### Discussion

If the current line join style is set to [kCALineJoinMiter](#) (page 173) (see [lineJoin](#) (page 170)), the miter limit determines whether the lines should be joined with a bevel instead of a miter. The length of the miter is divided by the line width. If the result is greater than the miter limit, the path is drawn with a bevel.

Default is 10.0.

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAShapeLayer.h

**path**

The path defining the shape to be rendered. Animatable.

@property CGPathRef path

**Discussion**

Defaults to `nil`.

**Availability**

Available in Mac OS X v10.6 and later.

**Related Sample Code**

CoreAnimationText

**Declared In**

CAShapeLayer.h

**strokeColor**

The color used to stroke the shape's path. Animatable.

@property CGColorRef strokeColor

**Discussion**

Setting `strokeColor` to `nil` results in no stroke being rendered.

Default is `nil`.

**Availability**

Available in Mac OS X v10.6 and later.

**Declared In**

CAShapeLayer.h

## Constants

### Shape Fill Mode Values

These constants specify the possible fill modes for [fillRule](#) (page 168).

```
NSString *const kCAFillRuleNonZero;
NSString *const kCAFillRuleEvenOdd;
```

**Constants**

`kCAFillRuleNonZero`

Specifies the non-zero winding rule. Count each left-to-right path as +1 and each right-to-left path as -1. If the sum of all crossings is 0, the point is outside the path. If the sum is nonzero, the point is inside the path and the region containing it is filled.

Available in Mac OS X v10.6 and later.

Declared in `CAShapeLayer.h`.

`kCAFillRuleEvenOdd`

Specifies the even-odd winding rule. Count the total number of path crossings. If the number of crossings is even, the point is outside the path. If the number of crossings is odd, the point is inside the path and the region containing it should be filled.

Available in Mac OS X v10.6 and later.

Declared in `CAShapeLayer.h`.

**Line Join Values**

These constants specify the shape of the joints between connected segments of a stroked path. Used by [lineJoin](#) (page 170). [Figure 17-2](#) (page 171) shows the appearance of the line join styles.

```
NSString *const kCALineJoinMiter;
NSString *const kCALineJoinRound;
NSString *const kCALineJoinBevel;
```

**Constants**

`kCALineJoinMiter`

Specifies a miter line shape of the joints between connected segments of a stroked path.

Available in Mac OS X v10.6 and later.

Declared in `CAShapeLayer.h`.

`kCALineJoinRound`

Specifies a round line shape of the joints between connected segments of a stroked path.

Available in Mac OS X v10.6 and later.

Declared in `CAShapeLayer.h`.

`kCALineJoinBevel`

Specifies a bevel line shape of the joints between connected segments of a stroked path.

Available in Mac OS X v10.6 and later.

Declared in `CAShapeLayer.h`.

**Line Cap Values**

These constants specify the shape of endpoints for an open path when stroked. Used by [lineCap](#) (page 169). [Figure 17-1](#) (page 169) shows the appearance of the line cap styles.

```
NSString *const kCALineCapButt;  
NSString *const kCALineCapRound;  
NSString *const kCALineCapSquare;
```

**Constants**

kCALineCapButt

Specifies a butt line cap style for endpoints for an open path when stroked.

Available in Mac OS X v10.6 and later.

Declared in `CAShapeLayer.h`.

kCALineCapRound

Specifies a round line cap style for endpoints for an open path when stroked.

Available in Mac OS X v10.6 and later.

Declared in `CAShapeLayer.h`.

kCALineCapSquare

Specifies a square line cap style for endpoints for an open path when stroked.

Available in Mac OS X v10.6 and later.

Declared in `CAShapeLayer.h`.

# CATextLayer Class Reference

---

<b>Inherits from</b>	CALayer : NSObject
<b>Conforms to</b>	NSCoding (CALayer) CAMediaTiming (CALayer) NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.5 and later.
<b>Declared in</b>	CATextLayer.h
<b>Companion guides</b>	Core Animation Programming Guide Core Animation Cookbook
<b>Related sample code</b>	CALayerEssentials CoreAnimationKioskStyleMenu GeekGameBoard

## 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 178) *property*  
The text to be rendered by the receiver.

### Text Visual Properties

`font` (page 177) *property*  
The font used to render the receiver's text.

`fontSize` (page 177) *property*  
The font size used to render the receiver's text. Animatable.

`foregroundColor` (page 178) *property*  
The color used to render the receiver's text. Animatable.

### Text Alignment and Truncation

`wrapped` (page 179) *property*  
Determines whether the text is wrapped to fit within the receiver's bounds.

`alignmentMode` (page 177) *property*  
Determines how individual lines of text are horizontally aligned within the receiver's bounds.

`truncationMode` (page 179) *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 180). Defaults to `kCAAlignmentNatural` (page 180).

### Availability

Available in Mac OS X v10.5 and later.

### Related Sample Code

GeekGameBoard

### Declared In

`CATextLayer.h`

## font

The font used to render the receiver's text.

```
@property CFontRef font
```

### Discussion

May be either a `CFontRef`, 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 178) property is not an `NSAttributedString`.

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

### Availability

Available in Mac OS X v10.5 and later.

### Related Sample Code

CoreAnimationKioskStyleMenu

GeekGameBoard

### 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 178) 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 Mac OS X v10.5 and later.

**Related Sample Code**

CoreAnimationKioskStyleMenu  
GeekGameBoard

**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 178) 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 Mac OS X v10.5 and later.

**Related Sample Code**

CALayerEssentials  
CoreAnimationKioskStyleMenu  
GeekGameBoard

**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 Mac OS X v10.5 and later.

**Related Sample Code**

CALayerEssentials

CoreAnimationKioskStyleMenu

GeekGameBoard

**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 179). Defaults to [kCATruncationNone](#) (page 180).

**Availability**

Available in Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Declared In**

CATextLayer.h

## Constants

### Truncation modes

These constants are used by the [truncationMode](#) (page 179) property.

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

**Constants**

kCATruncationNone

If the [wrapped](#) (page 179) property is YES, the text is wrapped to the receiver's bounds, otherwise the text is clipped to the receiver's bounds.

Available in Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

Declared in CATextLayer.h.

**Declared In**

CATextLayer.h

**Horizontal alignment modes**

These constants are used by the [alignmentMode](#) (page 177) 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 Mac OS X v10.5 and later.

Declared in CATextLayer.h.

kCAAlignmentLeft

Text is visually left aligned.

Available in Mac OS X v10.5 and later.

Declared in CATextLayer.h.

`kCAAAlignmentRight`

Text is visually right aligned.

Available in Mac OS X v10.5 and later.

Declared in `CATextLayer.h`.

`kCAAAlignmentCenter`

Text is visually center aligned.

Available in Mac OS X v10.5 and later.

Declared in `CATextLayer.h`.

`kCAAAlignmentJustified`

Text is justified.

Available in Mac OS X v10.5 and later.

Declared in `CATextLayer.h`.

**Declared In**

`CATextLayer.h`



# CATransaction Class Reference

---

<b>Inherits from</b>	NSObject
<b>Conforms to</b>	NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.5 and later.
<b>Declared in</b>	CATransaction.h
<b>Companion guides</b>	Core Animation Programming Guide Core Animation Cookbook
<b>Related sample code</b>	CoreAnimationText GeekGameBoard LightTable NineSlice

## 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 application sends the `CATransaction` class a `begin` (page 186) message before modifying the layer tree, and a `commit` (page 186) 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 186)  
Begin a new transaction for the current thread.
- + [commit](#) (page 186)  
Commit all changes made during the current transaction.
- + [flush](#) (page 188)  
Flushes any extant implicit transaction.

### Overriding Animation Duration and Timing

- + [animationDuration](#) (page 185)  
Returns the animation duration used by all animations within this transaction group.
- + [setAnimationDuration:](#) (page 188)  
Sets the animation duration used by all animations within this transaction group.
- + [animationTimingFunction](#) (page 185)  
Returns the timing function used for all animations within this transaction group.
- + [setAnimationTimingFunction:](#) (page 189)  
Sets the timing function used for all animations within this transaction group.

### Temporarily Disabling Property Animations

- + [disableActions](#) (page 187)  
Returns whether actions triggered as a result of property changes made within this transaction group are suppressed.
- + [setDisableActions:](#) (page 190)  
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 187)  
Returns the completion block object.
- + [setCompletionBlock:](#) (page 189)  
Sets the completion block object.



## Managing Concurrency

+ [lock](#) (page 188)

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

+ [unlock](#) (page 191)

Relinquishes a previously acquired transaction lock.

## Getting and Setting Transaction Properties

+ [setValue:forKey:](#) (page 190)

Sets the arbitrary keyed-data for the specified key.

+ [valueForKey:](#) (page 191)

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.

+ (CTimeInterval)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 191) value returned by the [kCATransactionAnimationDuration](#) (page 192) key.

#### Availability

Available in Mac OS X v10.6 and later.

#### See Also

+ [setAnimationDuration:](#) (page 188)

#### Related Sample Code

LightTable

#### 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 191) value returned by the `kCATransactionAnimationTimingFunction` (page 192) key.

**Availability**

Available in Mac OS X v10.6 and later.

**See Also**

+ `setAnimationTimingFunction:` (page 189)

**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 Mac OS X v10.5 and later.

**See Also**

+ `commit` (page 186)

+ `flush` (page 188)

**Related Sample Code**

GeekGameBoard

**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 Mac OS X v10.5 and later.

**See Also**

+ `begin` (page 186)

+ `flush` (page 188)

**Related Sample Code**

CoreAnimationText

GeekGameBoard

**Declared In**

CATransaction.h

**completionBlock**

Returns the completion block object.

`+ (void)completionBlock`**Discussion**See [setCompletionBlock:](#) (page 189) for a description of the role of the completion block object.**Availability**

Available in Mac OS X v10.6 and later.

**See Also**[+ completionBlock](#) (page 187)**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 191) value returned by the `kCATransactionDisableActions` (page 192) key.**Availability**

Available in Mac OS X v10.6 and later.

**See Also**[+ setDisableActions:](#) (page 190)**Related Sample Code**

LightTable

**Declared In**

CATransaction.h

## flush

Flushes any extant implicit transaction.

```
+ (void)flush
```

### Discussion

Delays the commit until any nested explicit transactions have completed.

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

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

### Availability

Available in Mac OS X v10.5 and later.

### See Also

+ [begin](#) (page 186)

+ [commit](#) (page 186)

### Related Sample Code

GeekGameBoard

### 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 that is read, modified, and set is correctly managed.

### Availability

Available in Mac OS X v10.6 and later.

### See Also

+ [unlock](#) (page 191)

### 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 191) value of the `kCATransactionAnimationDuration` (page 192) key.

**Availability**

Available in Mac OS X v10.6 and later.

**See Also**

+ [animationDuration](#) (page 185)

**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 191) value of the `kCATransactionAnimationTimingFunction` (page 192) key.

**Availability**

Available in Mac OS X v10.6 and later.

**See Also**

+ [animationTimingFunction](#) (page 185)

**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 Mac OS X v10.6 and later.

**See Also**

+ [completionBlock](#) (page 187)

**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 190) with an NSNumber containing a YES for the [kCATransactionDisableActions](#) (page 192) key.

**Availability**

Available in Mac OS X v10.6 and later.

**See Also**

+ [disableActions](#) (page 187)

**Related Sample Code**

CoreAnimationText

NineSlice

**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 Mac OS X v10.5 and later.

**Related Sample Code**

GeekGameBoard

LightTable

**Declared In**

CATransaction.h

**unlock**

Relinquishes a previously acquired transaction lock.

```
+ (void)unlock
```

**Availability**

Available in Mac OS X v10.6 and later.

**See Also**

+ [lock](#) (page 188)

**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 Mac OS X v10.5 and later.

**Declared In**

CATransaction.h

## Constants

### Transaction properties

These constants define the property keys used by `valueForKey:` (page 191) and `setValue:forKey:` (page 190).

```
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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.6 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 Mac OS X v10.6 and later.

Declared in `CATransaction.h`.

#### Declared In

`CATransaction.h`



# CATransformLayer Class Reference

---

<b>Inherits from</b>	CALayer : NSObject
<b>Conforms to</b>	NSCoding (CALayer) CAMediaTiming (CALayer) NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.6 and later.
<b>Declared in</b>	CATransformLayer.h
<b>Companion guides</b>	Core Animation Programming Guide Core Animation Cookbook

## Overview

`CATransformLayer` objects are used to create true 3D layer hierarchies, rather than the flattened hierarchy rendering model used by other `CALayer` classes.

Unlike normal layers, transform layers do not flatten their sublayers into the plane at  $Z=0$ . Due to this, they do not support many of the features of the `CALayer` class compositing model:

- Only the sublayers of a transform layer are rendered. The `CALayer` properties that are rendered by a layer are ignored, including: `backgroundColor`, `contents`, border style properties, stroke style properties, etc.
- The properties that assume 2D image processing are also ignored, including: `filters`, `backgroundFilters`, `compositingFilter`, `mask`, `masksToBounds`, and shadow style properties.
- The `opacity` property is applied to each sublayer individually, the transform layer does not form a compositing group.
- The `hitTest:` (page 194) method should never be called on a transform layer as they do not have a 2D coordinate space into which the point can be mapped.

## Tasks

### Hit Testing

- [hitTest:](#) (page 194)

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

## Instance Methods

### 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.

#### Discussion

Transform layers do not support hit-testing. As a result, you should not send `hitTest:` messages to transform layers.

# CATransition Class Reference

---

<b>Inherits from</b>	CAAnimation : NSObject
<b>Conforms to</b>	NSCoding (CAAnimation) NSCopying (CAAnimation) CAAction (CAAnimation) CAMediaTiming (CAAnimation) NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.5 and later.
<b>Declared in</b>	CAAnimation.h
<b>Companion guides</b>	Core Animation Programming Guide Core Animation Cookbook
<b>Related sample code</b>	CocoaSlides

## 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 197) *property*

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

`endProgress` (page 196) *property*

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

### Transition Properties

`type` (page 197) *property*

Specifies the predefined transition type.

[subtype](#) (page 197) *property*

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

## Custom Transition Filter

[filter](#) (page 196) *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 197), and not greater than 1.0. If `endProgress` is less than [startProgress](#) (page 197) the behavior is undefined. The default value is 1.0.

#### Availability

Available in Mac OS X v10.5 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 197) and [subtype](#) (page 197) 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 Mac OS X v10.5 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 Mac OS X v10.5 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 198). The default is `nil`.

This property is ignored if a custom transition is specified in the `filter` (page 196) property.

**Availability**

Available in Mac OS X v10.5 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 198). This property is ignored if a custom transition is specified in the `filter` (page 196) property. The default is `kCATransitionFade` (page 198).

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CAAnimation.h

## Constants

### Common Transition Types

These constants specify the transition types that can be used with the [type](#) (page 197) 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 Mac OS X v10.5 and later.

Declared in CAAnimation.h.

kCATransitionMoveIn

The layer's content slides into place over any existing content. The [“Common Transition Subtypes”](#) (page 198) are used with this transition.

Available in Mac OS X v10.5 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 198) are used with this transition.

Available in Mac OS X v10.5 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 198) are used with this transition.

Available in Mac OS X v10.5 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 197) 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 Mac OS X v10.5 and later.

Declared in `CAAnimation.h`.

`kCATransitionFromLeft`

The transition begins at the left side of the layer.

Available in Mac OS X v10.5 and later.

Declared in `CAAnimation.h`.

`kCATransitionFromTop`

The transition begins at the top of the layer.

Available in Mac OS X v10.5 and later.

Declared in `CAAnimation.h`.

`kCATransitionFromBottom`

The transition begins at the bottom of the layer.

Available in Mac OS X v10.5 and later.

Declared in `CAAnimation.h`.

**Declared In**

`CATransition.h`





# CAValueFunction Class Reference

---

<b>Inherits from</b>	NSObject
<b>Conforms to</b>	NSCoding NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.6 and later.
<b>Declared in</b>	CAValueFunction.h
<b>Companion guides</b>	Core Animation Programming Guide Core Animation Cookbook

## Overview

The `CAValueFunction.h` class provides a more flexible means of applying functions to property values during animation. A value function defines an optional transformation that is applied to the interpolated value before it is set in the presentation layer.

Using value transform functions, animations can effect the transform property of a layer using arbitrary transforms of each component (no normalization to 360°) and concatenate in the normally when multiple animations are applied at once.

You use a value transform function that rotates from 0° to 180° around the z-axis by creating a `CAValueTransform` function specifying the `kCAValueFunctionRotateZ` (page 203) and then creating an animation with a `fromValue` of 0, a `toValue` of `M_PI`, and set the animation's `valueTransform` property to the value transform instance.

## Tasks

### Getting Value Function Properties

`name` (page 202) *property*

Returns the name of the value function. (read-only)

## Creating and Initializing Value Functions

+ `functionWithName:` (page 202)

Returns the value function object identified by the name.

## Properties

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

### name

Returns the name of the value function. (read-only)

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

### Availability

Available in Mac OS X v10.6 and later.

### Declared In

CAValueFunction.h

## Class Methods

### functionWithName:

Returns the value function object identified by the name.

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

### Parameters

*name*

The name of the value function.

### Return Value

A new `CAValueFunction` instance with the value function specified by the name.

### Discussion

The possible values for *name* are specified in “Rotate Value Functions” (page 203), “Scale Value Functions” (page 203), and “Translate Functions” (page 204).

### Availability

Available in Mac OS X v10.6 and later.

### Declared In

CAValueFunction.h

## Constants

### Rotate Value Functions

Rotate value transform functions construct a 4x4 matrix that represents the corresponding rotation matrix.

```
NSString * const kCAValueFunctionRotateX;
NSString * const kCAValueFunctionRotateY;
NSString * const kCAValueFunctionRotateZ;
```

#### Constants

`kCAValueFunctionRotateX`

A value function that rotates by the input value, in radians, around the x-axis. This value function expects a single input value.

Available in Mac OS X v10.6 and later.

Declared in `CAValueFunction.h`.

`kCAValueFunctionRotateY`

A value function that rotates by the input value, in radians, around the y-axis. This value function expects a single input value.

Available in Mac OS X v10.6 and later.

Declared in `CAValueFunction.h`.

`kCAValueFunctionRotateZ`

A value function that rotates by the input value, in radians, around the z-axis. This value function expects a single input value.

Available in Mac OS X v10.6 and later.

Declared in `CAValueFunction.h`.

### Scale Value Functions

Scale value transform functions construct a 4x4 matrix that represents the corresponding scale matrix.

```
NSString * const kCAValueFunctionScale;
NSString * const kCAValueFunctionScaleX;
NSString * const kCAValueFunctionScaleY;
NSString * const kCAValueFunctionScaleZ;
```

#### Constants

`kCAValueFunctionScale`

A value function scales by the input value along all three axis. Animations using this value transform function must provide animation values in an `NSArray` of three `NSNumber` instances that specify the (x, y, z) scale values.

Available in Mac OS X v10.6 and later.

Declared in `CAValueFunction.h`.

`kCAValueFunctionScaleX`

A value function scales by the input value along the x-axis. Animations referencing this value transform function must provide a single animation value.

Available in Mac OS X v10.6 and later.

Declared in `CAValueFunction.h`.

`kCAValueFunctionScaleY`

A value function scales by the input value along the y-axis. Animations referencing this value function must provide a single animation value.

Available in Mac OS X v10.6 and later.

Declared in `CAValueFunction.h`.

`kCAValueFunctionScaleZ`

A value function that scales by the input value along the z-axis. Animations referencing this value function must provide a single animation value.

Available in Mac OS X v10.6 and later.

Declared in `CAValueFunction.h`.

## Translate Functions

Translate value transform functions construct a 4x4 matrix that represents the corresponding translate matrix.

```
NSString * const kCAValueFunctionTranslate;
NSString * const kCAValueFunctionTranslateX;
NSString * const kCAValueFunctionTranslateY;
NSString * const kCAValueFunctionTranslateZ;
```

### Constants

`kCAValueFunctionTranslate`

A value function that translates by the input values along all three axis. Animations using this value transform function must provide animation values in an `NSArray` of three `NSNumber` instances that specify the (x, y, z) translate values.

Available in Mac OS X v10.6 and later.

Declared in `CAValueFunction.h`.

`kCAValueFunctionTranslateX`

A value function translates by the input value along the x-axis. Animations referencing this value function must provide a single input value.

Available in Mac OS X v10.6 and later.

Declared in `CAValueFunction.h`.

`kCAValueFunctionTranslateY`

A value function translates by the input value along the y-axis. Animations referencing this value function must provide a single input value.

Available in Mac OS X v10.6 and later.

Declared in `CAValueFunction.h`.

`kCAValueFunctionTranslateZ`

A value function translates by the input value along the z-axis. Animations referencing this value function must provide a single input value.

Available in Mac OS X v10.6 and later.

Declared in `CAValueFunction.h`.



# CIFilter Core Animation Additions

---

<b>Inherits from</b>	NSObject
<b>Conforms to</b>	NSCoding NSCopying NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Declared in</b>	CACIFilterAdditions.h
<b>Companion guides</b>	Core Animation Programming Guide Core Animation Cookbook Core Image Programming Guide

## Overview

Core Animation adds two additional properties to the `CIFilter` class. These properties are accessible through key-value coding as well as the properties declared below.

## Tasks

### Naming Filter Instances

[name](#) (page 208) *property*

The name of the receiver.

### Enabling Filter Instances

[enabled](#) (page 208) *property*

Determines if the receiver is enabled. Animatable.

- [isEnabled](#) (page 208)

A synthesized accessor for the [enabled](#) (page 208) property.

## Properties

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

### enabled

Determines if the receiver is enabled. Animatable.

```
@property BOOL enabled
```

#### Discussion

The receiver is applied to its input when this property is set to YES. Default is YES.

#### Availability

Available in Mac OS X v10.5 and later.

#### Declared In

CACIFilterAdditions.h

### name

The name of the receiver.

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

#### Discussion

Default is nil. Each `CIFilter` instance can have an assigned name. The name is used to construct key paths to the filter’s attributes. For example, if a `CIFilter` instance has the name “myExposureFilter”, you refer to attributes of the filter using a key path such as “filters.myExposureFilter.inputEV”. Layer animations may also access filter attributes via these key paths.

#### Availability

Available in Mac OS X v10.5 and later.

#### Declared In

CACIFilterAdditions.h

## Instance Methods

### isEnabled

A synthesized accessor for the [enabled](#) (page 208) property.

```
- (BOOL)isEnabled
```

#### See Also

[@property enabled](#) (page 208)



# NSNumber Core Animation Additions

---

<b>Inherits from</b>	NSObject
<b>Conforms to</b>	NSCoding NSCopying NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Declared in</b>	QuartzCore/CATransform3D.h
<b>Companion guides</b>	Core Animation Programming Guide Core Animation Cookbook

## Overview

Core Animation adds two methods to the Foundation framework's `NSNumber` class to support `CATransform3D` structure values.

## Tasks

### Creating an NSNumber

- + `valueWithCATransform3D:` (page 209)  
Creates and returns an `NSNumber` object that contains a given `CATransform3D` structure.

### Accessing Data

- `CATransform3DValue` (page 210)  
Returns an `CATransform3D` structure representation of the receiver.

## Class Methods

### **valueWithCATransform3D:**

Creates and returns an `NSNumber` object that contains a given `CATransform3D` structure.

```
+ (NSNumber *)valueWithCATransform3D:(CATransform3D)aTransform
```

**Parameters**

*aTransform*

The value for the new object.

**Return Value**

A new `NSNumber` object that contains the value of *aTransform*.

**Availability**

Available in Mac OS X v10.5 and later.

**Related Sample Code**

`AnimatedTableView`

**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 Mac OS X v10.5 and later.

**Declared In**

`CATransform3D.h`

# QCCompositionLayer Class Reference

---

<b>Inherits from</b>	CAOpenGLLayer : CALayer : NSObject
<b>Conforms to</b>	QCCompositionRenderer NSCoding (CALayer) CAMediaTiming (CALayer) NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/Quartz.framework/Frameworks/QuartzComposer.framework
<b>Availability</b>	Available in Mac OS X v10.5 and later.
<b>Declared in</b>	QuartzComposer/QCCompositionLayer.h
<b>Companion guides</b>	Core Animation Programming Guide Quartz Composer Programming Guide
<b>Related sample code</b>	CALayerEssentials CocoaSlides CoreAnimationKioskStyleMenu

## Overview

The `QCCompositionLayer` class loads, plays, and controls Quartz Composer compositions in a Core Animation layer hierarchy. The composition tracks the Core Animation layer time and is rendered directly at the current dimensions of the `QCCompositionLayer` object.

An archived `QCCompositionLayer` object saves the composition that's loaded at the time the layer is archived. It detects layer usage and pauses or resumes the composition appropriately. A `QCCompositionLayer` object starts rendering the composition automatically when the layer is placed in a visible layer hierarchy. The layer stops rendering when it is hidden or removed from the visible layer hierarchy.

You can pass data to the input ports, or retrieve data from the output ports, of the root patch of a composition by accessing the `patch` attribute of the `QCCompositionLayer` instance using methods provided by the `QCCompositionRenderer` protocol.

**Note:** You must not modify the `asynchronous` property of the superclass `CAOpenGLLayer`.

## Tasks

### Creating the Layer

- + [compositionLayerWithFile:](#) (page 213)  
Creates and returns an instance of a composition layer using the Quartz Composer composition in the specified file.
- + [compositionLayerWithComposition:](#) (page 212)  
Creates and returns an instance of a composition layer using the provided Quartz Composer composition.
- [initWithFile:](#) (page 214)  
Initializes and returns a composition layer using the Quartz Composer composition in the specified file.
- [initWithComposition:](#) (page 214)  
Initializes and returns a composition layer using the provided Quartz Composer composition.

### Getting the Composition

- [composition](#) (page 213)  
Returns the composition associated with the layer.

## Class Methods

### **compositionLayerWithComposition:**

Creates and returns an instance of a composition layer using the provided Quartz Composer composition.

```
+ (QCCompositionLayer*)compositionLayerWithComposition:(QCComposition*)composition
```

#### **Parameters**

*composition*

The Quartz Composer composition to use as content.

#### **Return Value**

An autoreleased, initialized `QCCompositionLayer` object or `nil` if initialization is not successful.

#### **Availability**

Available in Mac OS X v10.5 and later.

#### **See Also**

+ [compositionLayerWithFile:](#) (page 213)

**Declared In**

QCCompositionLayer.h

**compositionLayerWithFile:**

Creates and returns an instance of a composition layer using the Quartz Composer composition in the specified file.

```
+ (QCCompositionLayer*)compositionLayerWithFile:(NSString*)path
```

**Parameters***path*

A string that specifies the location of a Quartz Composer composition.

**Return Value**

An autoreleased, initialized `QCCompositionLayer` object or `nil` if initialization is not successful.

**Availability**

Available in Mac OS X v10.5 and later.

**See Also**

+ [compositionLayerWithComposition:](#) (page 212)

**Related Sample Code**

CALayerEssentials

CocoaSlides

CoreAnimationKioskStyleMenu

**Declared In**

QCCompositionLayer.h

## Instance Methods

**composition**

Returns the composition associated with the layer.

```
- (QCComposition*) composition
```

**Return Value**

The composition object associated with the layer or `nil` if there is none.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

QCCompositionLayer.h

## initWithComposition:

Initializes and returns a composition layer using the provided Quartz Composer composition.

```
- (id)initWithComposition:(QCComposition*)composition
```

### Parameters

*composition*

The Quartz Composer composition to use as content.

### Return Value

The initialized `QCCompositionLayer` object or `nil` if initialization is not successful.

### Availability

Available in Mac OS X v10.5 and later.

### See Also

- [initWithFile:](#) (page 214)

### Declared In

`QCCompositionLayer.h`

## initWithFile:

Initializes and returns a composition layer using the Quartz Composer composition in the specified file.

```
- (id)initWithFile:(NSString*)path
```

### Parameters

*path*

A string that specifies the location of a Quartz Composer composition.

### Return Value

The initialized `QCCompositionLayer` object or `nil` if initialization is not successful.

### Availability

Available in Mac OS X v10.5 and later.

### See Also

- [initWithComposition:](#) (page 214)

### Declared In

`QCCompositionLayer.h`

# Protocols

---





# CAAction Protocol Reference

---

<b>Adopted by</b>	CAAnimation
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.5 and later.
<b>Declared in</b>	CALayer.h
<b>Companion guides</b>	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 217) message.

## Tasks

### Responding to an Action

- `runActionForKey:object:arguments:` (page 217) *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 Mac OS X v10.5 and later.

**Declared In**

`CALayer.h`

# CALayoutManager Protocol Reference

(informal protocol)

<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Declared in</b>	CALayer.h
<b>Companion guides</b>	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 126) or `layoutSublayers` (page 119).

## Tasks

### Layout Layers

- [invalidateLayoutOfLayer:](#) (page 219)  
Invalidates the layout of the specified layer.
- [layoutSublayersOfLayer:](#) (page 220)  
Layout each of the sublayers in the specified layer.

### Calculate Layer Size

- [preferredSizeOfLayer:](#) (page 220)  
Returns the preferred size of the specified layer in its coordinate system.

## Instance Methods

### **invalidateLayoutOfLayer:**

Invalidates the layout of the specified layer.

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

**Parameters**

*layer*

The layer that requires layout.

**Discussion**

This method is called when the preferred size of the specified layer may have changed. The receiver should invalidate any cached state.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CALayer.h

**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 Mac OS X v10.5 and later.

**Declared In**

CALayer.h

**preferredSizeOfLayer:**

Returns the preferred size of the specified layer in its coordinate system.

```
- (CGSize)preferredSizeOfLayer:(CALayer *)layer
```

**Parameters**

*layer*

The layer that requires layout.

**Return Value**

The preferred size of the layer in the coordinate space of *layer*.

**Discussion**

This method is called when the preferred size of the specified layer may have changed. The receiver is responsible for recomputing the preferred size and returning it. If this method is not implemented the preferred size is assumed to be the size of the bounds of *layer*.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CALayer.h



# CAMediaTiming Protocol Reference

---

<b>Adopted by</b>	CAAnimation CALayer
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in Mac OS X v10.5 and later.
<b>Declared in</b>	CAMediaTiming.h
<b>Companion guides</b>	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 232) 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 225) *required property*

Specifies the begin time of the receiver in relation to its parent object, if applicable. (required)

`timeOffset` (page 227) *required property*

Specifies an additional time offset in active local time. (required)

## Repeating Animations

`repeatCount` (page 226) *required property*

Determines the number of times the animation will repeat. (required)

`repeatDuration` (page 226) *required property*

Determines how many seconds the animation will repeat for. (required)

## Duration and Speed

`duration` (page 225) *required property*

Specifies the basic duration of the animation, in seconds. (required)

`speed` (page 226) *required property*

Specifies how time is mapped to receiver's time space from the parent time space. (required)

## Playback Modes

`autoreverses` (page 224) *required property*

Determines if the receiver plays in the reverse upon completion. (required)

`fillMode` (page 225) *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 Mac OS X v10.5 and later.

#### **Related Sample Code**

CoreAnimationKioskStyleMenu

#### **Declared In**

CAMediaTiming.h



## beginTime

Specifies the begin time of the receiver in relation to its parent object, if applicable. (required)

```
@property CTimeInterval beginTime
```

### Discussion

Defaults to 0.

### Availability

Available in Mac OS X v10.5 and later.

### Related Sample Code

Fireworks

### Declared In

CAMediaTiming.h

## duration

Specifies the basic duration of the animation, in seconds. (required)

```
@property CTimeInterval duration
```

### Discussion

Defaults to 0.

### Availability

Available in Mac OS X v10.5 and later.

### Related Sample Code

AnimatedTableView

CoreAnimationKioskStyleMenu

Fireworks

### 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 227). The default is `kCAFillModeRemoved` (page 227).

### Availability

Available in Mac OS X v10.5 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 226) and `repeatCount` (page 226) are specified the behavior is undefined.

Setting this property to `HUGE_VALF` will cause the animation to repeat forever.

### Availability

Available in Mac OS X v10.5 and later.

### Related Sample Code

CoreAnimationKioskStyleMenu

### 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 226) and `repeatCount` (page 226) are specified the behavior is undefined.

### Availability

Available in Mac OS X v10.5 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 Mac OS X v10.5 and later.

### Related Sample Code

Fireworks

### Declared In

CAMediaTiming.h

## timeOffset

Specifies an additional time offset in active local time. (required)

```
@property CTimeInterval timeOffset
```

### Discussion

Defaults to 0. .

### Availability

Available in Mac OS X v10.5 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 225) 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 Mac OS X v10.5 and later.

Declared in `CAMediaTiming.h`.

`kCAFillModeForwards`

The receiver remains visible in its final state when the animation is completed.

Available in Mac OS X v10.5 and later.

Declared in `CAMediaTiming.h`.

`kCAFillModeBackwards`

The receiver clamps values before zero to zero when the animation is completed.

Available in Mac OS X v10.5 and later.

Declared in `CAMediaTiming.h`.

`kCAFillModeBoth`

The receiver clamps values at both ends of the object's time space

Available in Mac OS X v10.5 and later.

Declared in `CAMediaTiming.h`.

kCAFillModeFrozen

The mode was deprecated before Mac OS X v10.5 shipped.

Deprecated in Mac OS X v10.5 and later.

Declared in CAMediaTiming.h.

**Declared In**

CAMediaTiming.h

# Other References

---



# Core Animation Function Reference

---

<b>Framework:</b>	QuartzCore/QuartzCore.h
<b>Declared in</b>	CABase.h CATransform3D.h

## Overview

## Functions by Task

### Timing Functions

[CACurrentMediaTime](#) (page 232)  
Returns the current absolute time, in seconds.

### Transform Functions

[CATransform3DIsIdentity](#) (page 233)  
Returns a Boolean value that indicates whether the transform is the identity transform.

[CATransform3DEqualToTransform](#) (page 232)  
Returns a Boolean value that indicates whether the two transforms are exactly equal.

[CATransform3DMakeTranslation](#) (page 235)  
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 234)  
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 234)  
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 235)  
Translate 't' by '(tx, ty, tz)' and return the result:  $t' = \text{translate}(tx, ty, tz) * t$ .

[CATransform3DScale](#) (page 235)  
Scale 't' by '(sx, sy, sz)' and return the result:  $t' = \text{scale}(sx, sy, sz) * t$ .

[CATransform3DRotate](#) (page 235)  
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:  $t' = \text{rotation}(\text{angle}, x, y, z) * t$ .

[CATransform3DConcat](#) (page 232)

Concatenate 'b' to 'a' and return the result:  $t' = a * b$ .

[CATransform3DInvert](#) (page 233)

Invert 't' and return the result. Returns the original matrix if 't' has no inverse.

[CATransform3DMakeAffineTransform](#) (page 234)

Return a transform with the same effect as affine transform 'm'.

[CATransform3DIsAffine](#) (page 233)

Returns true if 't' can be exactly represented by an affine transform.

[CATransform3DGetAffineTransform](#) (page 233)

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.

```
CTimeInterval CACurrentMediaTime (void);
```

#### Return Value

A `CFTimeInterval` derived by calling `mach_absolute_time()` and converting the result to seconds.

#### Availability

Available in Mac OS X v10.5 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 Mac OS X v10.5 and later.

#### Related Sample Code

`AnimatedTableView`

#### 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 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 Mac OS X v10.5 and later.

**Related Sample Code**

CALayerEssentials

Core Animation QuickTime Layer

**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 Mac OS X v10.5 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:  $t' = \text{rotation}(\text{angle}, x, y, z) * t$ .

```
CATransform3D CATransform3DRotate (CATransform3D t, CGFloat angle, CGFloat x,
CGFloat y, CGFloat z)
```

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CATransform3D.h

**CATransform3DScale**

Scale 't' by '(sx, sy, sz)' and return the result:  $t' = \text{scale}(sx, sy, sz) * t$ .

```
CATransform3D CATransform3DScale (CATransform3D t, CGFloat sx, CGFloat sy, CGFloat
sz)
```

**Availability**

Available in Mac OS X v10.5 and later.

**Related Sample Code**

AnimatedTableView

CoreAnimationText

**Declared In**

CATransform3D.h

**CATransform3DTranslate**

Translate 't' by '(tx, ty, tz)' and return the result:  $t' = \text{translate}(tx, ty, tz) * t$ .

```
CATransform3D CATransform3DTranslate (CATransform3D t, CGFloat tx, CGFloat ty,
CGFloat tz);
```

**Availability**

Available in Mac OS X v10.5 and later.

**Related Sample Code**

AnimatedTableView

CoreAnimationText

**Declared In**

CATransform3D.h

# Drawing

---

This chapter discusses drawing issues when using Core Animation and other technologies.

## Drawing Layer Content With Application Kit Classes

Core Animation `CALayer` class defines a delegate method, `drawLayer:inContext:`, that you can implement and draw your layer content using Quartz 2D drawing functions. However, Cocoa developers who have complete and working drawing solutions based on the Application Kit drawing classes may wish to continue using that code.

Listing 1 shows an implementation of the `CALayer` delegate method `drawLayer:inContext:` that creates an `NSGraphicsContext` from the `CGContextRef` passed as the `inContext:` parameter. Layer delegates can use this technique to display content created using `NSBezierPath`, `NSColor`, `NSImage` and other Application Kit classes.

**Listing 1** Drawing into a layer using Application Kit classes

```
- (void)drawLayer:(CALayer *)layer inContext:(CGContextRef)ctx
{
    NSGraphicsContext *nsGraphicsContext;
    nsGraphicsContext = [NSGraphicsContext graphicsContextWithGraphicsPort:ctx
                                                                flipped:NO];

    [NSGraphicsContext saveGraphicsState];
    [NSGraphicsContext setCurrentContext:nsGraphicsContext];

    // ...Draw content using NS APIs...
    NSRect aRect=NSMakeRect(10.0,10.0,30.0,30.0);
    NSBezierPath *thePath=[NSBezierPath bezierPathWithRect:aRect];
    [[NSColor redColor] set];
    [thePath fill];

    [NSGraphicsContext restoreGraphicsState];
}
```



# What Is Core Animation?

---

Core Animation is a collection of Objective-C classes for graphics rendering, projection, and animation. It provides fluid animations using advanced compositing effects while retaining a hierarchical layer abstraction that is familiar to developers using the Application Kit and Cocoa Touch view architectures.

Dynamic, animated user interfaces are hard to create, but Core Animation makes creating these interfaces easier by providing:

- High performance compositing with a simple approachable programming model.
- A familiar view-like abstraction that allows you to create complex user interfaces using a hierarchy of layer objects.
- A lightweight data structure. You can display and animate hundreds of layers simultaneously.
- An abstract animation interface that allows animations to run on a separate thread, independent of your application's run loop. Once an animation is configured and starts, Core Animation assumes full responsibility for running it at frame rate.
- Improved application performance. Applications need only redraw content when it changes. Minimal application interaction is required for resizing and providing layout services layers. Core Animation also eliminates application code that runs at the animation frame-rate.
- A flexible layout manager model, including a manager that allows the position and size of a layer to be set relative to attributes of sibling layers.

Using Core Animation, developers can create dynamic user interfaces for their applications without having to use low-level graphics APIs such as OpenGL to get respectable animation performance.

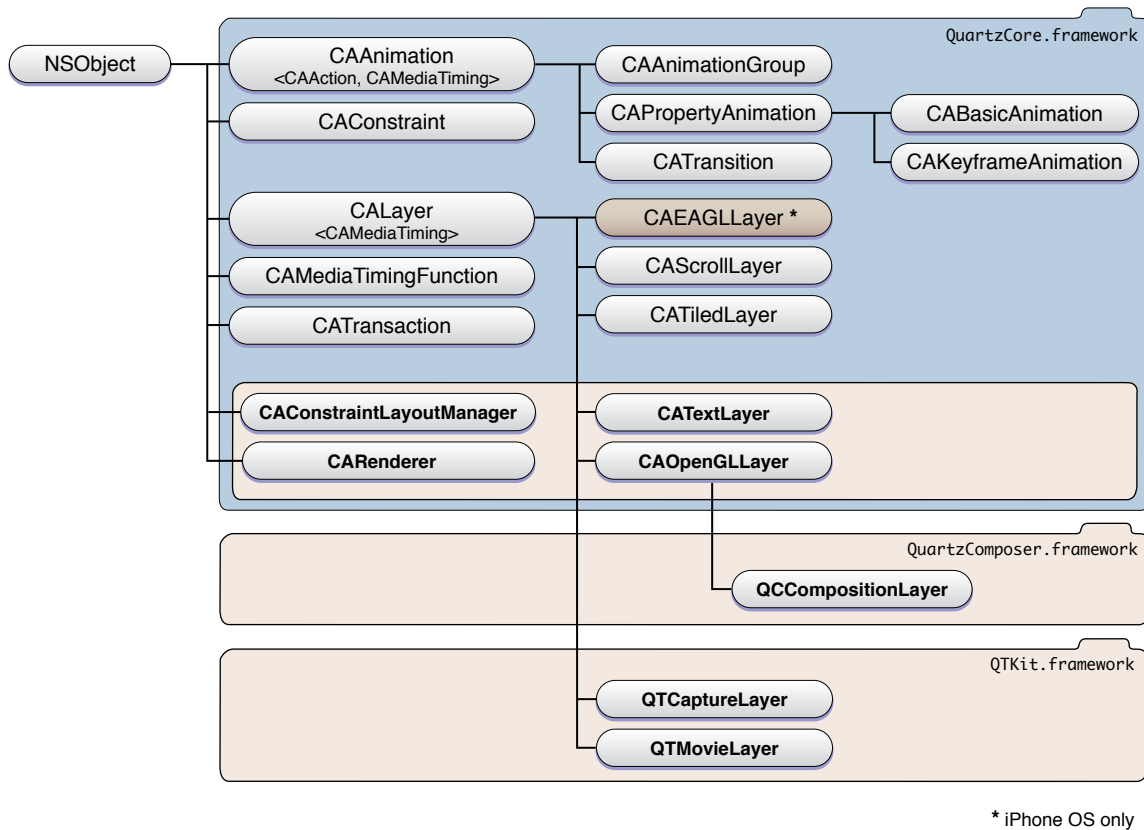
## Core Animation Classes

Core Animation classes can be grouped into several categories:

- Layer classes that provide content for display
- Animation and timing classes
- Layout and constraint classes
- A transaction class that groups multiple layer changes into an atomic update

The basic Core Animation classes are contained in the Quartz Core framework, although additional layer classes can be defined in other frameworks. “Core Animation Classes” shows the class hierarchy of Core Animation.

**Figure 1** Core Animation class hierarchy



## Layer Classes

The layer classes are the foundation of Core Animation and provide an abstraction that should be familiar to developers who have used `NSView` or `UIView`. Basic layer functionality is provided by the `CALayer` class, which is the parent class for all types of Core Animation layers.

As with an instance of a view class, a `CALayer` instance has a single parent layer (the superlayer) and a collection of sublayers, creating a hierarchy of layers that is referred to as the layer tree. Layers are drawn from back to front just like views and specify their geometry relative to their superlayer, creating a local coordinate system. However, layers allow a more complex visual display by incorporating transform matrices that allow you to rotate, skew, scale, and project the layer content. “[Layer Geometry and Transforms](#)” (page 249) discusses layer geometry and transforms in more detail.

`CALayer` diverges from the Application Kit and Cocoa Touch view classes in that it is not necessary to subclass `CALayer` in order to display content. The content displayed by a `CALayer` instance can be provided by:

- Setting the layer’s content property to a Core Graphics image representation directly, or through delegation.
- Providing a delegate that draws directly into a Core Graphics image context.
- Setting any of the number of visual style properties that all layer types have in common, for example, background colors, opacity, and masking. Mac OS X applications also have access to visual properties that make use of Core Image filters.



- Subclassing `CALayer` and implementing any of the above techniques in a more encapsulated manner.

“[Providing Layer Content](#)” (page 261) describes the available techniques for providing the content for a layer. The visual style properties and the order in which they are applied to the content of a layer is discussed in “[Layer Style Properties](#)” (page 287).

In addition to the `CALayer` class, the Core Animation class collection provides additional classes that allow applications to display other types of content. The available classes differ slightly between Mac OS X and iOS. The following classes are available on both Mac OS X and iOS:

- `CAScrollViewLayer` class is a subclass of `CALayer` that simplifies displaying a portion of a layer. The extent of the scrollable area of a `CAScrollViewLayer` object is defined by the layout of its sublayers. `CAScrollViewLayer` does not provide keyboard or mouse event-handling, nor does it provide visible scrollers.
- `CATiledLayer` allows the display of large and complex images in incremental stages.

Mac OS X provides these additional classes:

- `CATextLayer` is a convenience class that creates a layer's content from a string or attributed string.
- `CAOpenGLLayer` provides an OpenGL rendering environment. You must subclass this class to provide content using OpenGL. The content can be static or can be updated over time.
- `QCCompositionLayer` (provided by the Quartz Composer framework) animates a Quartz Composer composition as its content.
- `QTMovieLayer` and `QTCaptureLayer` (provided by the QTKit framework) provides playback of QuickTime movies and live video.

iOS adds the following class:

- `CAEAGLLayer` provides an OpenGL ES rendering environment.

The `CALayer` class introduces the concept of a **key-value coding compliant container class**—that is, a class that can store arbitrary values, using key-value coding compliant methods, without having to create a subclass. `CALayer` also extends the `NSKeyValueCoding` informal protocol, adding support for default key values and automatic object wrapping for the additional structure types (`CGPoint`, `CGSize`, `CGRect`, `CGAffineTransform` and `CATransform3D`) and provides access to many of the fields of those structures by key path.

`CALayer` also manages the animations and actions that are associated with a layer. Layers receive action triggers in response to layers being inserted and removed from the layer tree, modifications being made to layer properties, or explicit developer requests. These actions typically result in an animation occurring. See “[Animation](#)” (page 267) and “[Layer Actions](#)” (page 271) for more information.

## Animation and Timing Classes

---

Many of the visual properties of a layer are implicitly animatable. By simply changing the value of an animatable property the layer will automatically animate from the current value to the new value. For example, setting a layer's `hidden` property to `YES` triggers an animation that causes the layer to gradually fade away. Most animatable properties have an associated default animation which you can easily customize and replace. A complete list of the animatable properties and their default animations are listed in “[Animatable Properties](#)” (page 307).

Animatable properties can also be explicitly animated. To explicitly animate a property you create an instance of one of Core Animation's animation classes and specify the required visual effects. An explicit animation doesn't change the value of the property in the layer, it simply animates it in the display.

Core Animation provides animation classes that can animate the entire contents of a layer or selected attributes using both basic animation and key-frame animation. All Core Animation's animation classes descend from the abstract class `CAAnimation`. `CAAnimation` adopts the `CAAnimationTiming` protocol which provides the simple duration, speed, and repeat count for an animation. `CAAnimation` also adopts the `CAAction` protocol. This protocol provides a standardized means for starting an animation in response to an action triggered by a layer.

The animation classes also define a timing function that describes the pacing of the animation as a simple Bezier curve. For example, a linear timing function specifies that the animation's pace is even across its duration, while an ease-out timing function causes an animation to slow down as it nears its duration.

Core Animation provides a number of additional abstract and concrete animation classes:

- `CATransition` provides a transition effect that affects the entire layer's content. It fades, pushes, or reveals layer content when animating. The stock transition effects can be extended by providing your own custom Core Image filters.
- `CAAnimationGroup` allows an array of animation objects to be grouped together and run concurrently.
- `CAPropertyAnimation` is an abstract subclass that provides support for animating a layer property specified by a key path.
- `CABasicAnimation` provides simple interpolation for a layer property.
- `CAKeyframeAnimation` provides support for key frame animation. You specify the key path of the layer property to be animated, an array of values that represent the value at each stage of the animation, as well as arrays of key frame times and timing functions. As the animation runs, each value is set in turn using the specified interpolation.

These animation classes are used by both Core Animation and Cocoa Animation proxies. [“Animation”](#) (page 267) describes the classes as they pertain to Core Animation, *Animation Types and Timing Programming Guide* contains a more in-depth exploration of their capabilities.

## Layout Manager Classes

---

Application Kit view classes provide the classic "struts and springs" model of positioning layers relative to their superlayer. While layers support this model, Core Animation on Mac OS X also provides a more flexible layout manager mechanism that allows developers to write their own layout managers.

Core Animation's `CAConstraint` class is a layout manager that arranges sublayers using a set of constraints that you specify. Each constraint (encapsulated by instances of the `CAConstraint` class) describes the relationship of one geometric attribute of a layer (the left, right, top, or bottom edge or the horizontal or vertical center) in relation to a geometric attribute of one of its sibling layers or its superlayer.

Layout managers in general, and the constraint layout manager are discussed in [“Laying Out Core Animation Layers”](#) (page 279)

## Transaction Management Classes

---

Every modification to an animatable property of a layer must be part of a transaction. `CATransaction` is the Core Animation class responsible for batching multiple animation operations into atomic updates to the display. Nested transactions are supported.

Core Animation supports two types of transactions: implicit transactions and explicit transactions. Implicit transactions are created automatically when an animatable property of a layer 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 application sends the `CATransaction` class a `begin` message before modifying the layer, and a `commit` message afterwards.

Transaction management is discussed in [“Transactions”](#) (page 275).



# Timing

---

This chapter discusses timing issues when using Core Animation.

## Using a Single Timing Function For a Keyframe Animation

The `CAKeyframeAnimation` class provides a powerful means of animating layer properties. However, `CAKeyframeAnimation` does not allow you to specify a single animation timing function that is used for the entire path. Instead you are required to specify the timing using the `keyTimes` (page 76) property, or by specifying an array of timing functions in the `timingFunctions` (page 78) property.

You can provide a single timing function for the animation by grouping the keyframe animation in a `CAAnimationGroup`, and setting the group animation's timing function to the desired `CAMediaTimingFunction`. The animation group's timing function and duration take precedence over the keyframe animation's timing properties.

A code fragment that implements this strategy is shown in Listing 1.

### Listing 1 Using a single timing function for a keyframe animation

```
// create the path for the keyframe animation
CGMutablePathRef thePath = CGPathCreateMutable();
CGPathMoveToPoint(thePath, NULL, 15.0f, 15.0f);
CGPathAddCurveToPoint(thePath, NULL,
                      15.0f, 250.0f,
                      295.0f, 250.0f,
                      295.0f, 15.0f);

// create an explicit keyframe animation that
// animates the target layer's position property
// and set the animation's path property
CAKeyframeAnimation *theAnimation = [CAKeyframeAnimation
                                     animationWithKeyPath:@"position"];
theAnimation.path = thePath;

// create an animation group and add the keyframe animation
CAAnimationGroup *theGroup = [CAAnimationGroup animation];
theGroup.animations = [NSArray arrayWithObject:theAnimation];

// set the timing function for the group and the animation duration
theGroup.timingFunction = [CAMediaTimingFunction
                           functionWithName:kCAMediaTimingFunctionEaseIn];

theGroup.duration = 15.0;
// release the path
CFRelease(thePath);
```

## Timing

```
// adding the animation to the target layer causes it  
// to begin animating  
[theLayer addAnimation:theGroup forKey:@"animatePosition"];
```

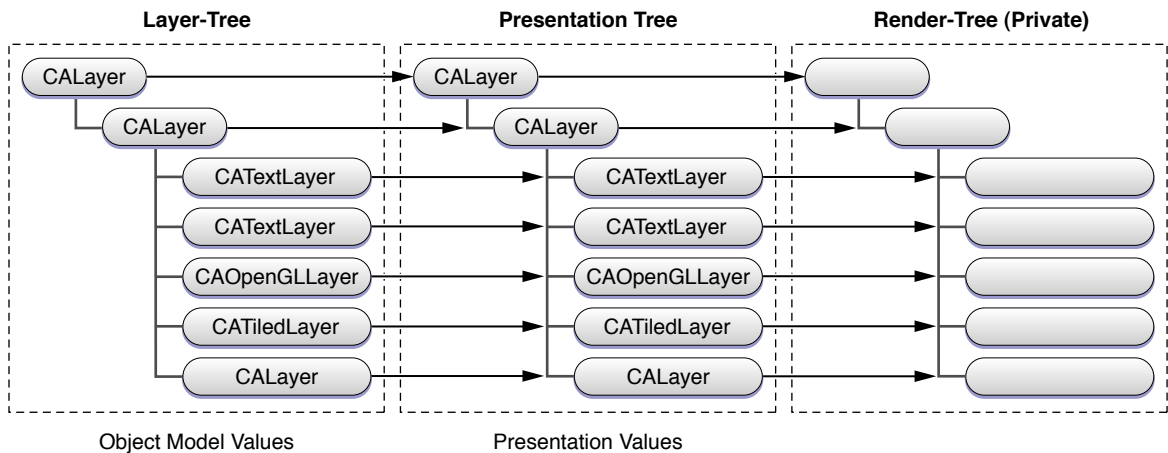
# Core Animation Rendering Architecture

While there are obvious similarities between Core Animation layers and Cocoa views the biggest conceptual divergence is that layers do not render directly to the screen.

Where `NSView` and `UIView` are clearly view objects in the model-view-controller design pattern, Core Animation layers are actually model objects. They encapsulate geometry, timing and visual properties, and they provide the content that is displayed, but the actual display is not the layer's responsibility.

Each visible layer tree is backed by two corresponding trees: a presentation tree and a render tree. Figure 1 shows an example layer-tree using the Core Animation layer classes available in Mac OS X.

**Figure 1** Core Animation Rendering Architecture



The layer tree contains the object model values for each layer. These are the values you set when you assign a value to a layer property.

The presentation tree contains the values that are currently being presented to the user as an animation takes place. For example, setting a new value for the `backgroundColor` of a layer immediately changes the value in the layer tree. However, the `backgroundColor` value in the corresponding layer in the presentation tree will be updated with the interpolated colors as they are displayed to the user.

The render-tree uses the value in the presentation-tree when rendering the layer. The render-tree is responsible for performing the compositing operations independent of application activity; rendering is done in a separate process or thread so that it has minimal impact on the application's run loop.

You can query an instance of `CALayer` for its corresponding presentation layer while an animation transaction is in process. This is most useful if you intend to change the current animation and want to begin the new animation from the currently displayed state.





# Layer Geometry and Transforms

---

This chapter describes the components of a layer's geometry, how they interrelate, and how transform matrices can produce complex visual effects.

## Layer Coordinate System

The layer's location and size are expressed using the same coordinate system that the Quartz graphics environment uses. By default, the graphics environment origin (0.0,0.0) is located in the lower left, and values are specified as floating-point numbers that increase up and to the right in coordinate system units. The coordinate system units, the unit square, is the size of a 1.0 by 1.0 rectangle.

Every layer instance defines and maintains its own coordinate system. All sublayers are positioned, and drawing is done, relative to this coordinate system. Methods are provided to convert points, rectangles and sizes from one layer coordinate system to another. A layer's coordinate system should be considered the base coordinate system for all the content of the layer, including its sublayers.

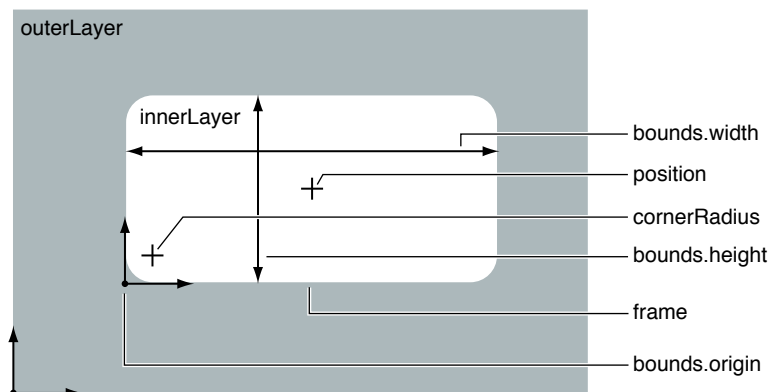
**iOS Note:** The default root layer of a `UIView` instance uses a flipped coordinate system that matches the default coordinate system of a `UIView` instance—the origin is in the top-left and values increase down and to the right. Layers created by instantiating `CALayer` directly use the standard Core Animation coordinate system.

## Specifying a Layer's Geometry

While layers and the layer-tree are analogous to Cocoa views and the view hierarchy in many ways, a layer's geometry is specified in a different, and often simpler, manner. All of a layer's geometric properties, including the layer's transform matrices, can be implicitly and explicitly animated.

Figure 1 shows the properties used to specify a layer's geometry in context.

Figure 1 CALayer geometry properties



The `position` property is a `CGPoint` that specifies the position of the layer relative to its superlayer, and is expressed in the superlayer's coordinate system.

The `bounds` property is a `CGRect` that provides the size of the layer (`bounds.size`) and the origin (`bounds.origin`). The bounds origin is used as the origin of the graphics context when you override a layer's drawing methods.

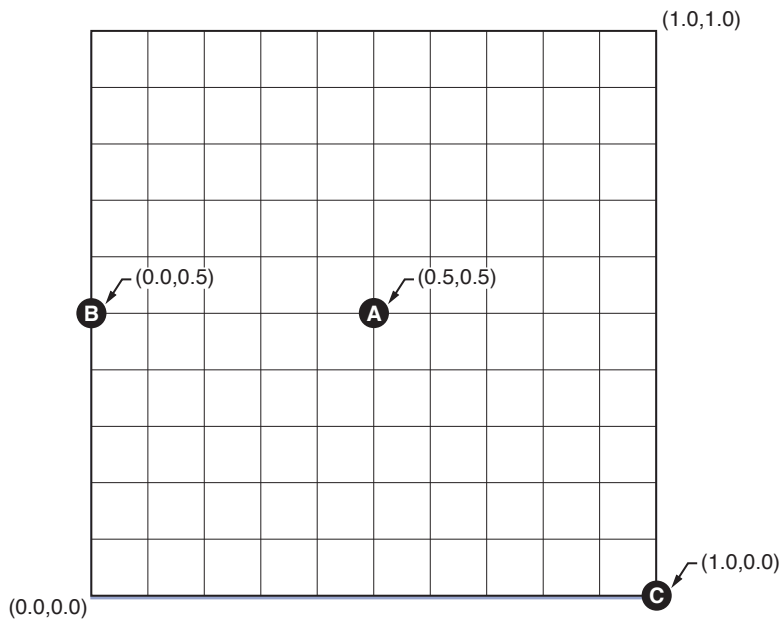
Layers have an implicit `frame` that is a function of the `position`, `bounds`, `anchorPoint`, and `transform` properties. Setting a new frame rectangle changes the layer's `position` and `bounds` properties appropriately, but the frame itself is not stored. When a new frame rectangle is specified the bounds origin is undisturbed, while the bounds size is set to the size of the frame. The layer's position is set to the proper location relative to the anchor point. When you get the `frame` property value, it is calculated relative to the `position`, `bounds`, and `anchorPoint` properties.

The `anchorPoint` property is a `CGPoint` that specifies a location within the bounds of a layer that corresponds with the `position` coordinate. The anchor point specifies how the bounds are positioned relative to the `position` property, as well as serving as the point that transforms are applied around. It is expressed in the unit coordinate system—the lower left of the layer bounds is 0.0,0.0, and the upper right is 1.0,1.0.

When you specify the frame of a layer, `position` is set relative to the anchor point. When you specify the position of the layer, `bounds` is set relative to the anchor point.

Figure 2 shows three example values for an anchor point.

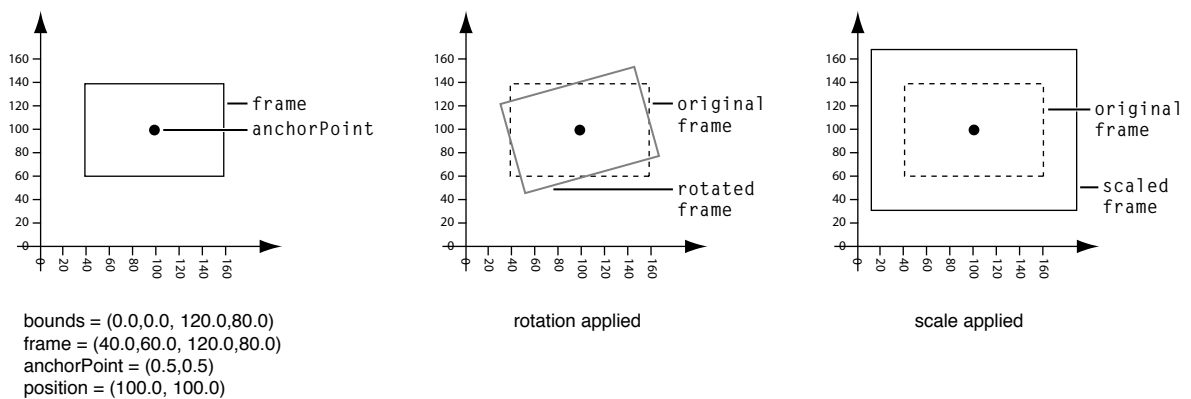
**Figure 2** Three anchorPoint values



The default value for `anchorPoint` is `(0.5,0.5)` which corresponds to the center of the layer's bounds (shown as point A in Figure 2.) Point B shows the position of an anchor point set to `(0.0,0.5)`. Finally, point C `(1.0,0.0)` causes specifies that the layer's `position` is set to the bottom right corner of the frame.

The relationship of the `frame`, `bounds`, `position`, and `anchorPoint` properties is shown in Figure 3.

**Figure 3** Layer Origin of `(0.5,0.5)`

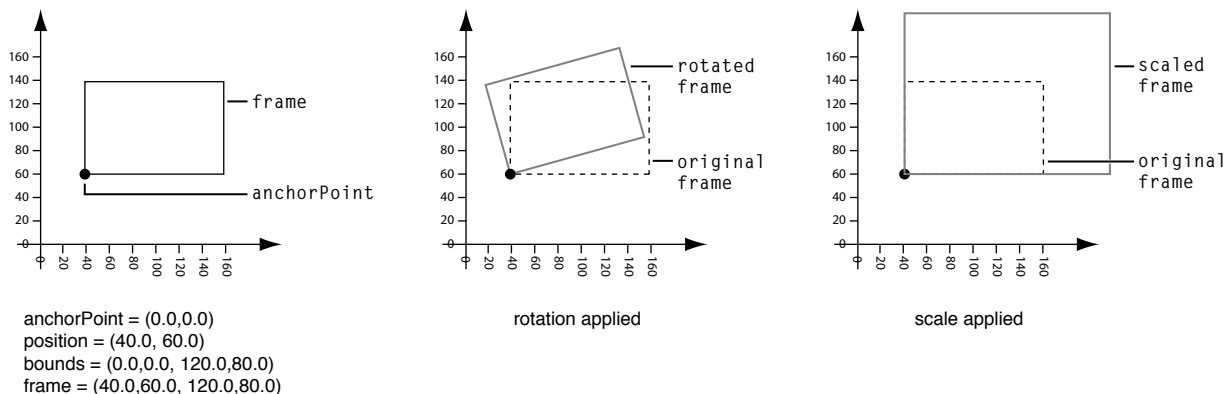


In this example the `anchorPoint` is set to the default value of `(0.5,0.5)`, which corresponds to the center of the layer. The `position` of the layer is set to `(100.0,100.0)`, and the `bounds` is set to the rectangle `(0.0, 0.0, 120.0, 80.0)`. This causes the `frame` property to be calculated as `(40.0, 60.0, 120.0, 80.0)`.

If you created a new layer, and set only the layer's `frame` property to `(40.0, 60.0, 120.0, 80.0)`, the `position` property would be automatically set to `(100.0,100.0)`, and the `bounds` property to `(0.0, 0.0, 120.0, 80.0)`.

Figure 4 shows a layer with the same `frame` rectangle as the layer in Figure 3. However, in this case the `anchorPoint` of the layer is set to `(0.0,0.0)`, which corresponds with the bottom left corner of the layer.

**Figure 4** Layer Origin of `(0.0,0.0)`



With the `frame` set to `(40.0, 60.0, 120.0, 80.0)`, the value of the `bounds` property is the same, but the value of the `position` property has changed.

Another aspect of layer geometry that differs from Cocoa views is that you can specify a radius that is used to round the corners of the layer. The `cornerRadius` property specifies a radius the layer uses when drawing content, clipping sublayers, and drawing the border and shadow.

The `zPosition` property specifies the z-axis component of the layer's position. The `zPosition` is intended to be used to set the visual position of the layer relative to its sibling layers. It should not be used to specify the order of layer siblings, instead reorder the layer in the sublayer array.

## Transforming a Layer's Geometry

Once established, you can transform a layer's geometry using matrix transformations. The `Transform` (page 133) data structure defines a homogenous three-dimensional transform (a 4 by 4 matrix of `CGFloat` values) that is used to rotate, scale, offset, skew, and apply perspective transformations to a layer.

Two layer properties specify transform matrices: `transform` and `sublayerTransform`. The matrix specified by the `transform` property is applied to the layer and its sublayers relative to the layer's `anchorPoint`. Figure 3 shows how rotation and scaling transforms affect a layer when using an `anchorPoint` of `(0.5,0.5)`, the default value. Figure 4 shows how the same transform matrices affect a layer when an `anchorPoint` of `(0.0,0.0)`. The matrix specified by the `sublayerTransform` property is applied only to the layer's sublayers, rather than to the layer itself.

You create and modify `CATransform3D` data structures in one of the following ways:

- using the `CATransform3D` functions
- modifying the data structure members directly
- using key-value coding and key paths.

The constant `CATransform3DIdentity` is the identity matrix, a matrix that has no scale, rotation, skewing, or perspective applied. Applying the identity matrix to a layer causes it to be displayed with its default geometry.

## Transform Functions

The transform functions available in Core Animation operate on matrices. You can use these functions (shown in Table 1) to construct a matrix that you later apply to a layer or its sublayers by modifying the transform or sublayerTransform properties respectively. The transform functions either operate on, or return, a `CATransform3D` data structure. This enables you to construct simple or complex transforms that you can readily reuse.

**Table 1** CATransform3D transform functions for translation, rotation, and scaling

Function	Use
<code>CATransform3DMakeTranslation</code> (page 235)	Returns a transform that translates by '(tx, ty, tz)'. $t' = [1\ 0\ 0; 0\ 1\ 0\ 0; 0\ 0\ 1\ 0; tx\ ty\ tz\ 1]$ .
<code>CATransform3DTranslate</code> (page 235)	Translate 't' by '(tx, ty, tz)' and return the result: $* t' = translate(tx, ty, tz) * t$ .
<code>CATransform3DMakeScale</code> (page 234)	Returns a transform that scales by '(sx, sy, sz)': $* t' = [sx\ 0\ 0\ 0; 0\ sy\ 0\ 0; 0\ 0\ sz\ 0; 0\ 0\ 0\ 1]$ .
<code>CATransform3DScale</code> (page 235)	Scale 't' by '(sx, sy, sz)' and return the result: $* t' = scale(sx, sy, sz) * t$ .
<code>CATransform3DMakeRotation</code> (page 234)	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.
<code>CATransform3DRotate</code> (page 235)	Rotate 't' by 'angle' radians about the vector '(x, y, z)' and return the result. $t' = rotation(angle, x, y, z) * t$ .

The angle of rotation is specified in radians rather than degrees. The following functions allow you to convert between radians and degrees.

```
CGFloat DegreesToRadians(CGFloat degrees) {return degrees * M_PI / 180;};
CGFloat RadiansToDegrees(CGFloat radians) {return radians * 180 / M_PI;};
```

Core Animation provides a transform function that inverts a matrix, `CATransform3DInvert`. Inversion is generally used to provide reverse transformation of points within transformed objects. Inversion can be useful when you need to recover a value that has been transformed by a matrix: invert the matrix, and multiply the value by the inverted matrix, and the result is the original value.

Functions are also provided that allow you to convert a `CATransform3D` matrix to a `CGAffineTransform` matrix, if the `CATransform3D` matrix can be expressed as such.

**Table 2** CATransform3D transform functions for CGAffineTransform conversion

Function	Use
<a href="#">CATransform3DMakeAffineTransform</a> (page 234)	Returns a CATransform3D with the same effect as the passed affine transform.
<a href="#">CATransform3DIsAffine</a> (page 233)	Returns YES if the passed CATransform3D can be exactly represented an affine transform.
<a href="#">CATransform3DGetAffineTransform</a> (page 233)	Returns the affine transform represented by the passed CATransform3D.

Functions are provided for comparing transform matrices for equality with the identity matrix, or another transform matrix.

**Table 3** CATransform3D transform functions for testing equality

Function	Use
<a href="#">CATransform3DIsIdentity</a> (page 233)	Returns YES if the transform is the identity transform.
<a href="#">CATransform3DEqualToTransform</a> (page 232)	Returns YES if the two transforms are exactly equal..

## Modifying the Transform Data Structure

---

You can modify the value of any of the CATransform3D data structure members as you would any other data structure. Listing 1 contains the definition of the CATransform3D data structure, the structure members are shown in their corresponding matrix positions.

**Listing 1** CATransform3D structure

```
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;
```

The example in Listing 2 illustrates how to configure a CATransform3D as a perspective transform.

**Listing 2** Modifying the CATransform3D data structure directly

```
CATransform3D aTransform = CATransform3DIdentity;
// the value of zDistance affects the sharpness of the transform.
zDistance = 850;
aTransform.m34 = 1.0 / -zDistance;
```

## Modifying a Transform Using Key Paths

Core Animation extends the key-value coding protocol to allow getting and setting of the common values of a layer's `CATransform3D` matrix through key paths. Table 4 describes the key paths for which a layer's `transform` and `sublayerTransform` properties are key-value coding and observing compliant.

**Table 4** CATransform3D key paths

Field Key Path	Description
<code>rotation.x</code>	The rotation, in radians, in the x axis.
<code>rotation.y</code>	The rotation, in radians, in the y axis.
<code>rotation.z</code>	The rotation, in radians, in the z axis.
<code>rotation</code>	The rotation, in radians, in the z axis. This is identical to setting the <code>rotation.z</code> field.
<code>scale.x</code>	Scale factor for the x axis.
<code>scale.y</code>	Scale factor for the y axis.
<code>scale.z</code>	Scale factor for the z axis.
<code>scale</code>	Average of all three scale factors.
<code>translation.x</code>	Translate in the x axis.
<code>translation.y</code>	Translate in the y axis.
<code>translation.z</code>	Translate in the z axis.
<code>translation</code>	Translate in the x and y axis. Value is an <code>NSSize</code> or <code>CGSize</code> .

You can not specify a structure field key path using Objective-C 2.0 properties. This will not work:

```
myLayer.transform.rotation.x=0;
```

Instead you must use `setValue:forKeyPath:` or `valueForKeyPath:` as shown below:

```
[myLayer setValue:[NSNumber numberWithInt:0] forKeyPath:@"transform.rotation.x"];
```





# Layer-Tree Hierarchy

---

Along with their own direct responsibilities for providing visual content and managing animations, layers also act as containers for other layers, creating a layer hierarchy.

This chapter describes the layer hierarchy and how you manipulate layers within that hierarchy.

## What Is a Layer-Tree Hierarchy?

The layer-tree is the Core Animation equivalent of the Cocoa view hierarchy. Just as an instance of `NSView` or `UIView` has superview and subviews, a Core Animation layer has a superlayer and sublayers. The layer-tree provides many of the same benefits as the view hierarchy:

- Complex interfaces can be assembled using simpler layers, avoiding monolithic and complex subclassing. Layers are well suited to this type of ‘stacking’ due to their complex compositing capabilities.
- Each layer declares its own coordinate system relative to its superlayer’s coordinate system. When a layer is transformed, its sublayers are transformed within it.
- A layer-tree is dynamic. It can be reconfigured as an application runs. Layers can be created, added as a sublayer first of one layer, then of another, and removed from the layer-tree.

## Displaying Layers in Views

Core Animation doesn’t provide a means for actually displaying layers in a window, they must be hosted by a view. When paired with a view, the view must provide event-handling for the underlying layers, while the layers provide display of the content.

The view system in iOS is built directly on top of Core Animation layers. Every instance of `UIView` automatically creates an instance of a `CALayer` class and sets it as the value of the view’s `layer` property. To display custom layer content in a `UIView` instance you simply add the layers as sublayers of the view’s layer.

On Mac OS X you must configure an `NSView` instance in such a way that it can host a layer. To display the root layer of a layer tree, you set a view’s layer and then configure the view to use layers as shown in Table 2.

### Listing 1 Inserting a layer into a view

```
// theView is an existing view in a window
// theRootLayer is the root layer of a layer tree

[theView setLayer: theRootLayer];
[theView setWantsLayer:YES];
```

## Adding and Removing Layers from a Hierarchy

Simply instantiating a layer instance doesn't insert it into a layer-tree. Instead you add, insert, replace, and remove layers from the layer-tree using the methods described in .Table 1.

**Table 1** Layer-tree management methods.

Method	Result
<code>addSublayer:</code>	Appends the layer to the receiver's sublayers array.
<code>insertSublayer: atIndex:</code>	Inserts the layer as a sublayer of the receiver at the specified index.
<code>insertSublayer: below:</code>	Inserts the layer into the receiver's sublayers array, below the specified sublayer.
<code>insertSublayer: above:</code>	Inserts the layer into the receiver's sublayers array, above the specified sublayer.
<code>removeFromSuperlayer</code>	Removes the receiver from the sublayers array or mask property of the receiver's superlayer.
<code>replaceSublayer: with:</code>	Replaces the layer in the receiver's sublayers array with the specified new layer.

You can also set the sublayers of a layer using an array of layers, and setting the intended superlayer's sublayers property. When setting the sublayers property to an array populated with layer objects you must ensure that the layers have had their superlayer set to `nil`.

By default, inserting and removing layers from a visible layer-tree triggers an animation. When a layer is added as a sublayer the animation returned by the parent layer for the action identifier `kCAOnOrderIn` is triggered. When a layer is removed from a layer's sublayers the animation returned by the parent layer for the action identifier `kCAOnOrderOut` is triggered. Replacing a layer in a sublayer causes the animation returned by the parent layer for the action identifier `kCATransition` to be triggered. You can disable animation while manipulating the layer-tree, or alter the animation used for any of the action identifiers.

## Repositioning and Resizing Layers

After a layer has been created, you can move and resize it programmatically simply by changing the value of the layer's geometry properties: `frame`, `bounds`, `position`, `anchorPoint`, or `zPosition`.

If a layer's `needsDisplayOnBoundsChange` property is YES, the layer's content is recached when the layer's bounds changes. By default the `needsDisplayOnBoundsChange` property is no.

By default, setting the `frame`, `bounds`, `position`, `anchorPoint`, and `zPosition` properties causes the layer to animate the new values.

## Autoresizing Layers

`CALayer` provides a mechanism for automatically moving and resizing sublayers in response to their superlayer being moved or resized. In many cases simply configuring the autoresizing mask for a layer provides the appropriate behavior for an application.

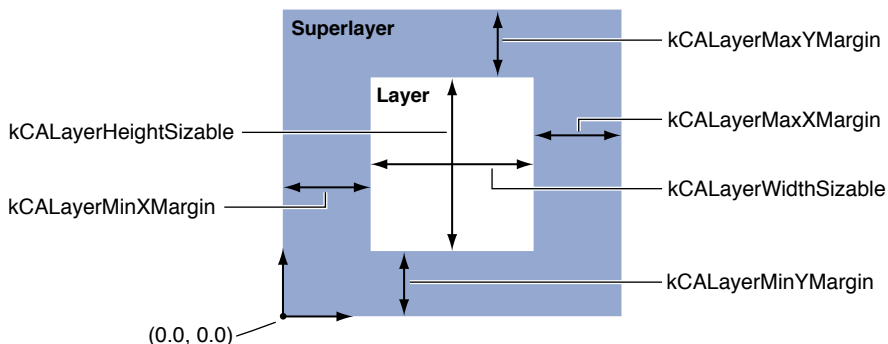
A layer's autoresizing mask is specified by combining the `CAAutoresizingMask` constants using the bitwise OR operator and the layer's `autoresizingMask` property to the resulting value. Table 2 shows each mask constant and how it effects the layer's resizing behavior.

**Table 2** Autoresizing mask values and descriptions

Autoresizing Mask	Description
<code>kCALayerHeightSizable</code>	If set, the layer's height changes proportionally to the change in the superlayer's height. Otherwise, the layer's height does not change relative to the superlayer's height.
<code>kCALayerWidthSizable</code>	If set, the layer's width changes proportionally to the change in the superlayer's width. Otherwise, the layer's width does not change relative to the superlayer's width.
<code>kCALayerMinXMargin</code>	If set, the layer's left edge is repositioned proportionally to the change in the superlayer's width. Otherwise, the layer's left edge remains in the same position relative to the superlayer's left edge.
<code>kCALayerMaxXMargin</code>	If set, the layer's right edge is repositioned proportionally to the change in the superlayer's width. Otherwise, the layer's right edge remains in the same position relative to the superlayer.
<code>kCALayerMaxYMargin</code>	If set, the layer's top edge is repositioned proportionally to the change in the superlayer's height. Otherwise, the layer's top edge remains in the same position relative to the superlayer.
<code>kCALayerMinYMargin</code>	If set, the layer's bottom edge is repositioned proportional to the change in the superlayer's height. Otherwise, the layer's bottom edge remains in the same position relative to the superlayer.

For example, to keep a layer in the lower-left corner of its superlayer, you use the mask `kCALayerMaxXMargin | kCALayerMaxYMargin`. When more than one aspect along an axis is made flexible, the resize amount is distributed evenly among them. Figure 1 provides a graphical representation of the position of the constant values.

**Figure 1** Layer autoresizing mask constants



When one of these constants is omitted, the layer's layout is fixed in that aspect; when a constant is included in the mask the layer's layout is flexible in that aspect.

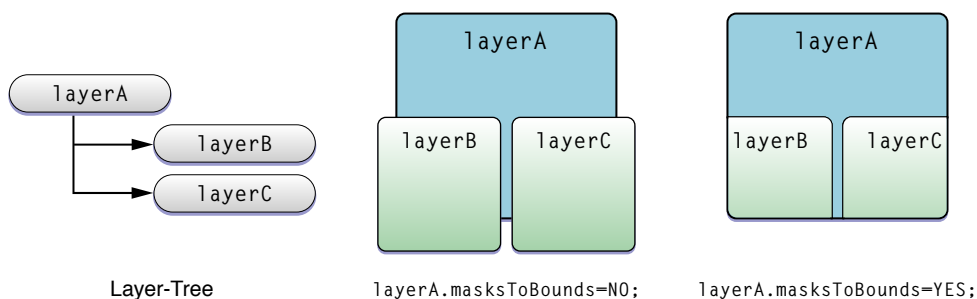
A subclass can override the `CALayer` methods `resizeSublayersWithOldSize:` and `resizeWithOldSuperlayerSize:` to customize the autoresizing behavior for a layer. A layer's `resizeSublayersWithOldSize:` method is invoked automatically by a layer whenever bounds property changes, and sends a `resizeWithOldSuperlayerSize:` message to each sublayer. Each sublayer compares the old bounds size to the new size and adjusts its position and size according to its autoresize mask.

## Clipping Sublayers

When subviews of a Cocoa view lie outside of the parent view's bounds, the views are clipped to the parent view. Layers remove this limitation, allowing sublayers to be displayed in their entirety, regardless of their position relative to the parent layer.

The value of a layer's `masksToBounds` property determines if sublayers are clipped to the parent. The default value of the `masksToBounds` property is `NO`, which prevents sublayers from being clipped to the parent. Figure 2 shows the results of setting the `masksToBounds` for `layerA` and how it will affect the display of `layerB` and `layerC`.

**Figure 2** Example Values of the `masksToBounds` property



# Providing Layer Content

---

When using Cocoa views you must subclass `NSView` or `UIView` and implement `drawRect:` in order to display anything. However `CALayer` instances can often be used directly, without requiring subclassing. Because `CALayer` is a key-value coding compliant container class, that is you can store arbitrary values in any instance, subclassing can often be avoided entirely.

## Providing CALayer Content

You specify the content of a `CALayer` instance in one of the following ways:

- Explicitly set the `contents` property of a layer instance using a `CGImageRef` that contains the content image.
- Specify a delegate that provides, or draws, the content.
- Subclass `CALayer` and override one of the display methods.

## Setting the Contents Property

---

A layer's content image is specified by `contents` property to a `CGImageRef`. This can be done from another object when the layer is created (as shown in Table 3) or at any other time.

### Listing 1 Setting a layer's contents property

```
CALayer *theLayer;

// create the layer and set the bounds and position
theLayer=[CALayer layer];
theLayer.position=CGPointMake(50.0f,50.0f);
theLayer.bounds=CGRectMake(0.0f,0.0f,100.0f,100.0f);

// set the contents property to a CGImageRef
// specified by theImage (loaded elsewhere)
theLayer.contents=theImage;
```

## Using a Delegate to Provide Content

---

You can draw content for your layer, or better encapsulate setting the layer's content image by creating a delegate class that implements one of the following methods: `displayLayer:` or `drawLayer:inContext:`.

Implementing a delegate method to draw the content does not automatically cause the layer to draw using that implementation. Instead, you must explicitly tell a layer instance to re-cache the content, either by sending it a `setNeedsDisplay` or `setNeedsDisplayInRect:` message, or by setting its `needsDisplayOnBoundsChange` property to YES.

Delegates that implement the `displayLayer:` method can determine which image should be displayed for the specified layer, and then set that layer's `contents` property accordingly. The example in implementation of `displayLayer:` in "Layer Coordinate System" sets the `contents` property of the layer depending on the value of the state key. Subclassing is not required to store the state value, as the `CALayer` instance acts as a key-value coding container.

**Listing 2** Example implementation of the delegate method `displayLayer:`

```
- (void)displayLayer:(CALayer *)theLayer
{
    // check the value of the layer's state key
    if ([[theLayer valueForKey:@"state"] boolValue])
    {
        // display the yes image
        theLayer.contents=[someHelperObject loadImage];
    }
    else {
        // display the no image
        theLayer.contents=[someHelperObject loadImage];
    }
}
```

If you must draw the layer's content rather than loading it from an image, you implement the `drawLayer:inContext:` delegate method. The delegate is passed the layer for which content is required and a `CGContextRef` to draw the content in.

The example in implementation of `drawLayer:inContext::` in "Specifying a Layer's Geometry" draws a path in using the `lineWidth` key value returned by the layer.

**Listing 3** Example implementation of the delegate method `drawLayer:inContext:`

```
- (void)drawLayer:(CALayer *)theLayer
    inContext:(CGContextRef)theContext
{
    CGMutablePathRef thePath = CGPathCreateMutable();

    CGContextMoveToPoint(thePath, NULL, 15.0f, 15.0f);
    CGContextAddCurveToPoint(thePath,
                            NULL,
                            15.0f, 250.0f,
                            295.0f, 250.0f,
                            295.0f, 15.0f);

    CGContextBeginPath(theContext);
    CGContextAddPath(theContext, thePath );

    CGContextSetLineWidth(theContext,
                          [[theLayer valueForKey:@"lineWidth"] floatValue]);
    CGContextStrokePath(theContext);
}
```

```

    // release the path
    CFRelease(thePath);
}

```

## Providing CALayer Content by Subclassing

---

Although often unnecessary, you can subclass `CALayer` and override the drawing and display methods directly. This is typically done when your layer requires custom behavior that can't be provided through delegation.

A subclass can override the `CALayer` display method and set the layer's contents to the appropriate image. The example in "Transforming a Layer's Geometry" provides the same functionality as the delegate implementation of `displayLayer:` in "Layer Coordinate System." The difference is that the subclass defines state as instance property, rather than depending on the key-value coding container ability of `CALayer`.

### Listing 4 Example override of the CALayer display method

```

- (void)display
{
    // check the value of the layer's state key
    if (self.state)
    {
        // display the yes image
        self.contents=[someHelperObject loadImage];
    }
    else {
        // display the no image
        self.contents=[someHelperObject loadImage];
    }
}

```

`CALayer` subclasses can draw the layer's content into a graphics context by overriding `drawInContext:`. The example in "Modifying the Transform Data Structure" produces the same content image as the delegate implementation in "Specifying a Layer's Geometry." Again, the only difference in the implementation is that `lineWidth` and `lineColor` are now declared as instance properties of the subclass.

### Listing 5 Example override of the CALayer drawInContext: method

```

- (void)drawInContext:(CGContextRef)theContext
{
    CGMutablePathRef thePath = CGPathCreateMutable();

    CGContextMoveToPoint(thePath, NULL, 15.0f, 15.0f);
    CGContextAddCurveToPoint(thePath,
                             NULL,
                             15.0f, 250.0f,
                             295.0f, 250.0f,
                             295.0f, 15.0f);

    CGContextBeginPath(theContext);
    CGContextAddPath(theContext, thePath );

    CGContextSetLineWidth(theContext,

```

```

        self.lineWidth);
CGContextSetStrokeColorWithColor(theContext,
                                self.lineColor);
CGContextStrokePath(theContext);
CFRelease(thePath);
}

```

Subclassing `CALayer` and implementing one of the drawing methods does not automatically cause drawing to occur. You must explicitly cause the instance to re-cache the content, either by sending it a `setNeedsDisplay` or `setNeedsDisplayInRect:` message, or by setting its `needsDisplayOnBoundsChange` property to `YES`.

## Positioning Content Within a Layer

The `CALayer` property `contentsGravity` allows you to position and scale the layer's contents image within the layer bounds. By default, the content image fills the layer's bounds entirely, ignoring the natural aspect ratio of the image.

Using the `contentsGravity` positioning constants you can specify that the image is placed along any of the layer's edges, in the layer's corners, or centered within the layer's bounds. However, when using the positioning constants the `contentsCenter` property is not used. [Table 1](#) (page 264) lists the positioning constants and their corresponding positions.

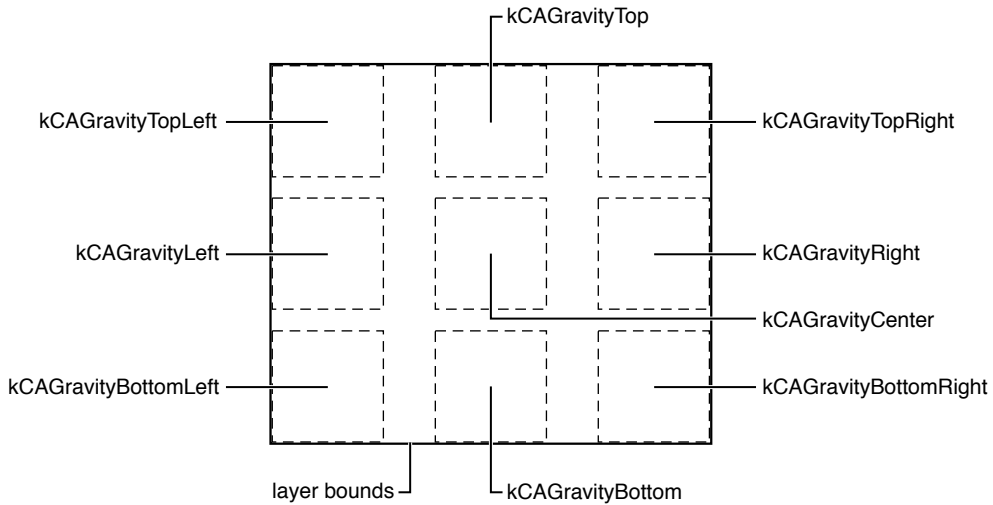
**Table 1** Positioning constants for a layer's `contentsGravity` property

Position constant	Description
<code>kCAGravityTopLeft</code>	Positions the content image in the top left corner of the layer.
<code>kCAGravityTop</code>	Positions the content image horizontally centered along the top edge of the layer.
<code>kCAGravityTopRight</code>	Positions the content image in the top right corner of the layer.
<code>kCAGravityLeft</code>	Positions the content image vertically centered on the left edge of the layer.
<code>kCAGravityCenter</code>	Positions the content image at the center of the layer.
<code>kCAGravityRight</code>	Positions the content image vertically centered on the right edge of the layer.
<code>kCAGravityBottomLeft</code>	Positions the content image in the bottom left corner of the layer.
<code>kCAGravityBottom</code>	Positions the content image centered along the bottom edge of the layer.
<code>kCAGravityBottomRight</code>	Positions the content image in the top right corner of the layer.

“Layer Coordinate System” shows the supported content positions and their corresponding constants.



**Figure 1** Position constants for a layer's contentsGravity property



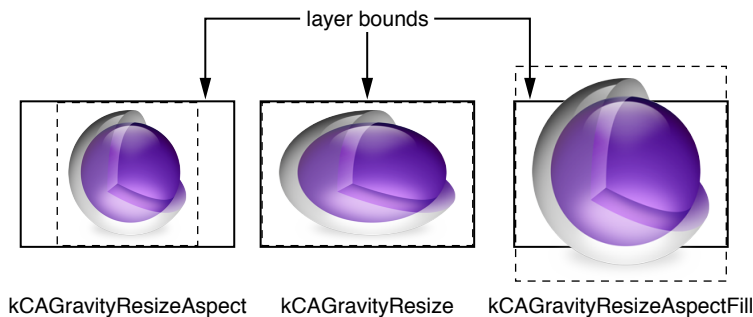
The content image can be scaled up, or down, by setting the `contentsGravity` property to one of the constants listed in [Table 2](#) (page 265). It is only when using one of these resize constants that the `contentsCenter` property affects the content image.

**Table 2** Scaling Constants For A Layer's `contentsGravity` Property

Scaling constant	Description
<code>kCAGravityResize</code>	Resize the content image to completely fill the layer bounds, potentially ignoring the natural aspect of the content. This is the default.
<code>kCAGravityResizeAspect</code>	Resize the content image to scale such that it is displayed as large as possible within the layer bounds, yet still retains its natural aspect.
<code>kCAGravityResizeAspectFill</code>	Resize the content image to scale such that it is displayed filling the layer bounds, yet retaining its natural aspect. This may cause the content to extend outside the layer bounds.

“Transforming a Layer’s Geometry” illustrates how a square image is resized to fit within a rectangular layer bounds using the resizing modes.

**Figure 2** Scaling constants for a layer's contentsGravity property



**Note:** Using any of the constants `kCAGravityResize`, `kCAGravityResizeAspect`, and `kCAGravityResizeAspectFill` make the gravity position constants in [Table 1](#) (page 264) irrelevant. The content will fill the layer bounds, so it is impossible to position the content using those constants.

# Animation

---

Animation is a key element of today's user interfaces. When using Core Animation animation is completely automatic. There are no animation loops or timers. Your application is not responsible for frame by frame drawing, or tracking the current state of your animation. The animation occurs automatically in a separate thread, without further interaction with your application.

This chapter provides an overview of the animation classes, and describes how to create both implicit and explicit animations.

## Animation Classes and Timing

Core Animation provides an expressive set of animation classes you can use in your application:

- `CABasicAnimation` provides simple interpolation between values for a layer property.
- `CAKeyframeAnimation` provides support for key frame animation. You specify the key path of the layer property to be animated, an array of values that represent the value at each stage of the animation, as well as arrays of key frame times and timing functions. As the animation runs, each value is set in turn using the specified interpolation.
- `CATransition` provides a transition effect that affects the entire layer's content. It fades, pushes, or reveals layer content when animating. The stock transition effects can be extended by providing your own custom Core Image filters.
- `CAAnimationGroup` allows an array of animation objects to be grouped together and run concurrently.

In addition to specifying the type of animation to perform, you must also specify the duration of the animation, the pacing (how the interpolated values are distributed across the duration), if the animation is to repeat and how many times, whether it should automatically reverse when each cycle is completed, and its visual state when the animation is completed. The animation classes and the `CAMediaTiming` protocol provides all this functionality and more.

`CAAnimation` and its subclasses and the timing protocols are shared by both Core Animation and the Cocoa Animation Proxy functionality. The classes are described in detail in *Animation Types and Timing Programming Guide*.

## Implicit Animation

Core Animation's implicit animation model assumes that all changes to animatable layer properties should be gradual and asynchronous. Dynamically animated scenes can be achieved without ever explicitly animating layers. Changing the value of an animatable layer property causes the layer to implicitly animate the change from the old value to the new value. While an animation is in-flight, setting a new target value causes the animation transition to the new target value from its current state.

Listing 1 shows how simple it is to trigger an implicit animation that animates a layer from its current position to a new position.

**Listing 1** Implicitly animating a layer’s position property

```
// assume that the layer is current positioned at (100.0,100.0)
theLayer.position=CGPointMake(500.0,500.0);
```

You can implicitly animate a single layer property at a time, or many. You can also implicitly animate several layers simultaneously. The code in Listing 2 causes four implicit animations to occur simultaneously.

**Listing 2** Implicitly animating multiple properties of multiple layers

```
// animate theLayer's opacity to 0 while moving it
// further away in the layer
theLayer.opacity=0.0;
theLayer.zPosition=-100;

// animate anotherLayer's opacity to 1
// while moving it closer in the layer
anotherLayer.opacity=1.0;
anotherLayer.zPosition=100.0;
```

Implicit animations use the duration specified in the default animation for the property, unless the duration has been overridden in an implicit or explicit transaction. See [“Overriding the Duration of Implied Animations”](#) (page 276) for more information.

## Explicit Animation

Core Animation also supports an explicit animation model. The explicit animation model requires that you create an animation object, and set start and end values. An explicit animation won’t start until you apply the animation to a layer. The code fragment in Listing 3 creates an explicit animation that transitions a layer’s opacity from fully opaque to fully transparent, and back over a 3 second duration. The animation doesn’t begin until it is added to the layer.

**Listing 3** Explicit animation

```
CABasicAnimation *theAnimation;

theAnimation=[CABasicAnimation animationWithKeyPath:@"opacity"];
theAnimation.duration=3.0;
theAnimation.repeatCount=2;
theAnimation.autoreverses=YES;
theAnimation.fromValue=[NSNumber numberWithInt:1.0];
theAnimation.toValue=[NSNumber numberWithInt:0.0];
[theLayer addAnimation:theAnimation forKey:@"animateOpacity"];
```

Explicit animations are especially useful when creating animations that run continuously. Listing 4 shows how to create an explicit animation that applies a CoreImage bloom filter to a layer, animating its intensity. This causes the “selection layer” to pulse, drawing the user’s attention.

**Listing 4** Continuous explicit animation example

```

// The selection layer will pulse continuously.
// This is accomplished by setting a bloom filter on the layer

// create the filter and set its default values
CIFilter *filter = [CIFilter filterWithName:@"CIBloom"];
[filter setDefaults];
[filter setValue:[NSNumber numberWithFloat:5.0] forKey:@"inputRadius"];

// name the filter so we can use the keypath to animate the inputIntensity
// attribute of the filter
[filter setName:@"pulseFilter"];

// set the filter to the selection layer's filters
[selectionLayer setFilters:[NSArray arrayWithObject:filter]];

// create the animation that will handle the pulsing.
CABasicAnimation* pulseAnimation = [CABasicAnimation animation];

// the attribute we want to animate is the inputIntensity
// of the pulseFilter
pulseAnimation.keyPath = @"filters.pulseFilter.inputIntensity";

// we want it to animate from the value 0 to 1
pulseAnimation.fromValue = [NSNumber numberWithFloat: 0.0];
pulseAnimation.toValue = [NSNumber numberWithFloat: 1.5];

// over a one second duration, and run an infinite
// number of times
pulseAnimation.duration = 1.0;
pulseAnimation.repeatCount = HUGE_VALF;

// we want it to fade on, and fade off, so it needs to
// automatically autoreverse.. this causes the intensity
// input to go from 0 to 1 to 0
pulseAnimation.autoreverses = YES;

// use a timing curve of easy in, easy out..
pulseAnimation.timingFunction = [CAMediaTimingFunction functionWithName:
kCAMediaTimingFunctionEaseInEaseOut];

// add the animation to the selection layer. This causes
// it to begin animating. We'll use pulseAnimation as the
// animation key name
[selectionLayer addAnimation:pulseAnimation forKey:@"pulseAnimation"];

```

## Starting and Stopping Explicit Animations

You start an explicit animation by sending a `addAnimation:forKey:` message to the target layer, passing the animation and an identifier as parameters. Once added to the target layer the explicit animation will run until the animation completes, or it is removed from the layer. The identifier used to add an animation to a layer is also used to stop it by invoking `removeAnimationForKey:.` You can stop all animations for a layer by sending the layer a `removeAllAnimations` message.



# Layer Actions

Layer actions are triggered in response to: layers being inserted and removed from the layer-tree, the value of layer properties being modified, or explicit application requests. Typically, action triggers result in an animation being displayed.

## The Role of Action Objects

An action object is an object that responds to an action identifier via the `CAAction` protocol. Action identifiers are named using standard dot-separated key paths. A layer is responsible for mapping action identifiers to the appropriate action object. When the action object for the identifier is located that object is sent the message defined by the `CAAction` protocol.

The `CALayer` class provides default action objects—instances of `CAAnimation`, a `CAAction` protocol compliant class—for all animatable layer properties. `CALayer` also defines the following action triggers that are not linked directly to properties, as well as the action identifiers in Table 1.

**Table 1** Action triggers and their corresponding identifiers

Trigger	Action identifier
A layer is inserted into a visible layer-tree, or the <code>hidden</code> property is set to <code>NO</code> .	The action identifier constant <code>kCAOnOrderIn</code> .
A layer is removed from a visible layer-tree, or the <code>hidden</code> property is set to <code>YES</code> .	The action identifier constant <code>kCAOnOrderOut</code> .
A layer replaces an existing layer in a visible layer tree using <code>replaceSublayer:with:</code> .	The action identifier constant <code>kCATransition</code> .

## Defined Search Pattern for Action Keys

When an action trigger occurs, the layer's `actionForKey:` method is invoked. This method returns an action object that corresponds to the action identifier passed as the parameter, or `nil` if no action object exists.

When the `CALayer` implementation of `actionForKey:` is invoked for an identifier the following search pattern is used:

1. If the layer has a delegate, and it implements the method `actionForLayer:forKey:` it is invoked, passing the layer, and the action identifier as parameters. The delegate's `actionForLayer:forKey:` implementation should respond as follows:
  - Return an action object that corresponds to the action identifier.

- Return `nil` if it doesn't handle the action identifier.
  - Return `NSNull` if it doesn't handle the action identifier and the search should be terminated.
2. The layer's `actions` dictionary is searched for an object that corresponds to the action identifier.
  3. The layer's `style` property is searched for an `actions` dictionary that contains the identifier.
  4. The layer's class is sent a `defaultActionForKey:` message. It will return an action object corresponding to the identifier, or `nil` if not found.

## Adopting the CAAction Protocol

The `CAAction` protocol defines how action objects are invoked. Classes that implement the `CAAction` protocol have a method with the signature `runActionForKey:object:arguments:`.

When the action object receives the `runActionForKey:object:arguments:` message it is passed the action identifier, the layer on which the action should occur, and an optional dictionary of parameters.

Typically, action objects are an instance of a `CAAnimation` subclass, which implements the `CAAction` protocol. You can, however, return an instance of any class that implements the protocol. When that instance receives the `runActionForKey:object:arguments:` message it should respond by performing its action.

When an instance of `CAAnimation` receives the `runActionForKey:object:arguments:` message it responds by adding itself to the layer's animations, causing the animation to run (see [Listing 1](#) (page 272)).

**Listing 1** `runActionForKey:object:arguments:` implementation that initiates an animation

```
- (void)runActionForKey:(NSString *)key
    object:(id)anObject
    arguments:(NSDictionary *)dict
{
    [[CALayer *)anObject addAnimation:self forKey:key];
}
```

## Overriding an Implied Animation

You can provide a different implied animation for an action identifier by inserting an instance of `CAAnimation` into the `actions` dictionary, into an `actions` dictionary in the `style` dictionary, by implementing the delegate method `actionForLayer:forKey:`, or subclassing a layer class, overriding `defaultActionForKey:` and returning the appropriate action object.

The example in [Listing 2](#) replaces the default implied animation for the `contents` property using delegation.

**Listing 2** Implied animation for the `contents` property

```
- (id<CAAction>)actionForLayer:(CALayer *)theLayer
```



```

        forKey:(NSString *)theKey
    {
        CATransition *theAnimation=nil;

        if ([theKey isEqualToString:@"contents"])
        {
            theAnimation = [[CATransition alloc] init];
            theAnimation.duration = 1.0;
            theAnimation.timingFunction = [CAMediaTimingFunction
functionWithName:kCAMediaTimingFunctionEaseIn];
            theAnimation.type = kCATransitionPush;
            theAnimation.subtype = kCATransitionFromRight;
        }

        return theAnimation;
    }

```

The example in [Listing 3](#) (page 273) disables the default animation for the `sublayers` property using the actions dictionary pattern.

**Listing 3**      Implied animation for the `sublayers` property

```

// get a mutable version of the current actions dictionary
NSMutableDictionary *customActions=[NSMutableDictionary
dictionaryWithDictionary:[theLayer actions]];

// add the new action for sublayers
[customActions setObject:[NSNull null] forKey:@"sublayers"];

// set theLayer actions to the updated dictionary
theLayer.actions=customActions;

```

## Temporarily Disabling Actions

By default, any time you change an animatable property, the appropriate animation will occur.

You can temporarily disable actions when modifying layer properties by using transactions. See [“Temporarily Disabling Layer Actions”](#) (page 275) for more information.



# Transactions

---

Every modification to a layer is part of a transaction. `CATransaction` is the Core Animation class responsible for batching multiple layer-tree modifications into atomic updates to the render tree.

This chapter describes the two types of transactions Core Animation supports: implicit transactions and explicit transactions.

## Implicit 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.

The example in Listing 1 modifies a layer's `opacity`, `zPosition`, and `position` properties, relying on the implicit transaction to ensure that the resulting animations occur at the same time.

### Listing 1 Animation using an implicit transaction

```
theLayer.opacity=0.0;
theLayer.zPosition=-200;
theLayer.position=CGPointMake(0.0,0.0);
```

**Important:** When modifying layer properties from threads that don't have a runloop, you must use explicit transactions.

## Explicit Transactions

You create an explicit transaction by sending the `CATransaction` class a `begin` message before modifying the layer tree, and a `commit` message afterwards. Explicit transactions are particularly useful when setting the properties of many layers at the same time (for example, while laying out multiple layers), temporarily disabling layer actions, or temporarily changing the duration of resulting implied animations.

## Temporarily Disabling Layer Actions

---

You can temporarily disable layer actions when changing layer property values by setting the value of the transaction's `kCATransactionDisableActions` to `true`. Any changes made during the scope of that transaction will not result in an animation occurring. Listing 2 shows an example that disables the fade animation that occurs when removing a `layer` from a visible layer-tree.

**Listing 2** Temporarily disabling a layer's actions

```
[CATransaction begin];
[CATransaction setValue:(id)kCFBooleanTrue
                      forKey:kCATransactionDisableActions];
[Layer removeFromSuperlayer];
[CATransaction commit];
```

## Overriding the Duration of Implied Animations

---

You can temporarily alter the duration of animations that run in response to changing layer property values by setting the value of the transaction's `kCATransactionAnimationDuration` key to a new duration. Any resulting animations in that transaction scope will use that duration rather than their own. Listing 3 shows an example that causes an animation to occur over 10 seconds rather than the duration specified by the `zPosition` and `opacity` animations..

**Listing 3** Overriding the animation duration

```
[CATransaction begin];
[CATransaction setValue:[NSNumber numberWithInt:10.0f]
                      forKey:kCATransactionAnimationDuration];
theLayer.zPosition=200.0;
theLayer.opacity=0.0;
[CATransaction commit];
```

Although the above example shows the duration bracketed by an explicit transaction `begin` and `commit`, you could omit those and use the implicit transaction instead.

## Nesting Transactions

---

Explicit transactions can be nested, allowing you to disable actions for one part of an animation, or using different durations for the implicit animations of properties that are modified. Only when the outer-most transaction is committed will the animations occur.

Listing 4 shows an example of nesting two transactions. The outer transaction sets the implied animation duration to 2 seconds and sets the layer's `position` property. The inner transaction sets the implied animation duration to 5 seconds and changes the layer's `opacity` and `zPosition`.

**Listing 4** Nesting explicit transactions

```
[CATransaction begin]; // outer transaction

// change the animation duration to 2 seconds
[CATransaction setValue:[NSNumber numberWithInt:2.0f]
                      forKey:kCATransactionAnimationDuration];
// move the layer to a new position
theLayer.position = CGPointMake(0.0,0.0);

[CATransaction begin]; // inner transaction
// change the animation duration to 5 seconds
[CATransaction setValue:[NSNumber numberWithInt:5.0f]
                      forKey:kCATransactionAnimationDuration];
```

## Transactions

```
// change the zPosition and opacity
theLayer.zPosition=200.0;
theLayer.opacity=0.0;

[CATransaction commit]; // inner transaction

[CATransaction commit]; // outer transaction
```



# Laying Out Core Animation Layers

`NSView` provides the classic "struts and springs" model of repositioning views relative to their superlayer when it resizes. While layers support this model, Core Animation on Mac OS X provides a more general layout manager mechanism that allows developers to write their own layout managers. A custom layout manager (which implements the `CALayoutManager` protocol) can be specified for a layer, which then assumes responsibility for providing layout of the layer's sublayers.

This chapter describes the constraints layout manager and how to configure a set of constraints.

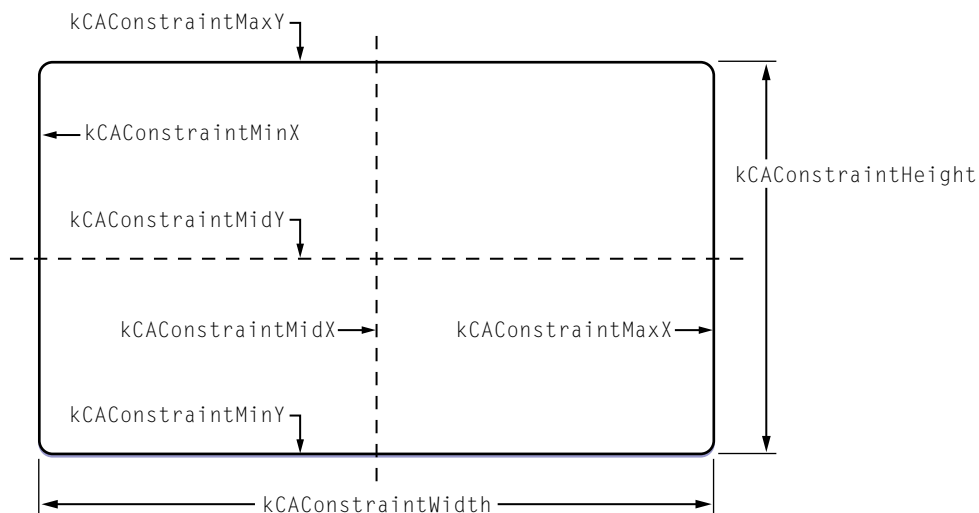
**iOS Note:** The `CALayer` class in iOS only supports the "struts and springs" positioning model, it does not provide custom layout managers.

## Constraints Layout Manager

Constraint-based layout allows you to specify the position and size of a layer using relationships between itself its sibling layers or its superlayer. The relationships are represented by instances of the `CAConstraint` class that are stored in an array in the sublayers' `constraints` property.

Figure 1 shows the layout attributes you can use when specifying relationships.

**Figure 1** Constraint layout manager attributes



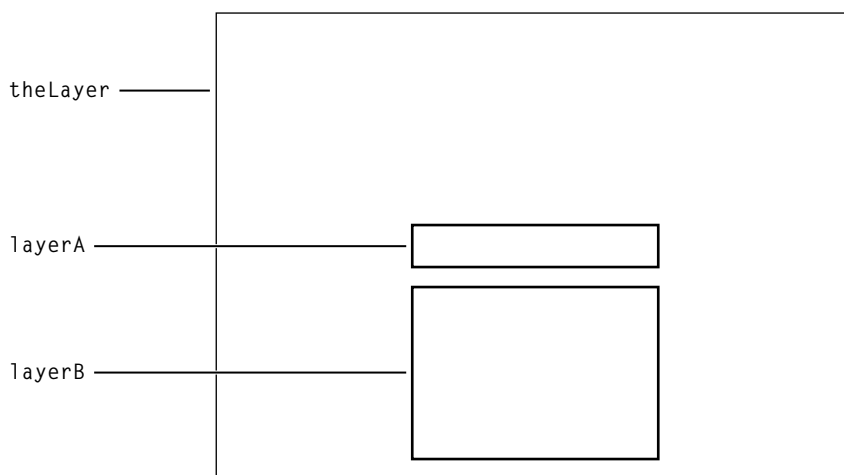
When using constraints layout you first create an instance of `CAConstraintLayoutManager` and set it as the parent layer's layout manager. You then create constraints for the the sublayers by instantiating `CAConstraint` objects and adding them to the sublayer's constraints using `addConstraint:`. Each `CAConstraint` instance encapsulates one geometry relationship between two layers on the same axis.

Sibling layers are referenced by name, using the `name` property of a layer. The special name `superlayer` is used to refer to the layer's superlayer.

A maximum of two relationships must be specified per axis. If you specify constraints for the left and right edges of a layer, the width will vary. If you specify constraints for the left edge and the width, the right edge of the layer will move relative to the superlayer's frame. Often you'll specify only a single edge constraint, the layer's size in the same axis will be used as the second relationship.

The example code in Listing 1 creates a layer, and then adds sublayers that are positioned using constraints. Figure 2 shows the resulting layout.

**Figure 2** Example constraints based layout



**Listing 1** Configuring a layer's constraints

```
// create and set a constraint layout manager for theLayer
theLayer.layoutManager=[CAConstraintLayoutManager layoutManager];

CALayer *layerA = [CALayer layer];
layerA.name = @"layerA";

layerA.bounds = CGRectMake(0.0,0.0,100.0,25.0);
layerA.borderWidth = 2.0;

[layerA addConstraint:[CAConstraint constraintWithAttribute:kCAConstraintMidY
                                                         relativeTo:@"superlayer"
                                                         attribute:kCAConstraintMidY]];

[layerA addConstraint:[CAConstraint constraintWithAttribute:kCAConstraintMidX
                                                         relativeTo:@"superlayer"
                                                         attribute:kCAConstraintMidX]];

[theLayer addSublayer:layerA];
```



```

CALayer *layerB = [CALayer layer];
layerB.name = @"layerB";
layerB.borderWidth = 2.0;

[layerB addConstraint:[CAConstraint constraintWithAttribute:kCAConstraintWidth
                                                         relativeTo:@"layerA"
                                                         attribute:kCAConstraintWidth]];

[layerB addConstraint:[CAConstraint constraintWithAttribute:kCAConstraintMidX
                                                         relativeTo:@"layerA"
                                                         attribute:kCAConstraintMidX]];

[layerB addConstraint:[CAConstraint constraintWithAttribute:kCAConstraintMaxY
                                                         relativeTo:@"layerA"
                                                         attribute:kCAConstraintMinY
                                                         offset:-10.0]];

[layerB addConstraint:[CAConstraint constraintWithAttribute:kCAConstraintMinY
                                                         relativeTo:@"superlayer"
                                                         attribute:kCAConstraintMinY
                                                         offset:+10.0]];

[theLayer addSublayer:layerB];

```

Here's what the code does:

1. Creates an instance of `CAConstraintLayoutManager` and sets it as the `layoutManager` property of `theLayer`.
2. Creates an instance of `CALayer` (`layerA`) and sets the layer's `name` property to "layerA".
3. The bounds of `layerA` is set to a `(0.0,0.0,100.0,25.0)`.
4. Creates a `CAConstraint` object, and adds it as a constraint of `layerA`.  
This constraint aligns the horizontal center of `layerA` with the horizontal center of the superlayer.
5. Creates a second `CAConstraint` object, and adds it as a constraint of `layerA`.  
This constraint aligns the vertical center of `layerA` with the vertical center of the superlayer.
6. Adds `layerA` as a sublayer of `theLayer`.
7. Creates an instance of `CALayer` (`layerB`) and sets the layer's `name` property to "layerB".
8. Creates a `CAConstraint` object, and adds it as a constraint of `layerA`.  
This constraint sets the width of `layerB` to the width of `layerA`.
9. Creates a second `CAConstraint` object, and adds it as a constraint of `layerB`.  
This constraint sets the horizontal center of `layerB` to be the same as the horizontal center of `layerA`.
10. Creates a third `CAConstraint` object, and adds it as a constraint of `layerB`.  
This constraint sets the top edge of `layerB` 10 points below the bottom edge of `layerA`.

11. Creates a fourth `CAConstraint` object, and adds it as a constraint of `layerB`.

This constraint sets the bottom edge of `layerB` 10 points above the bottom edge of the superlayer.



**Warning:** It is possible to create constraints that result in circular references to the same attributes. In cases where the layout is unable to be computed, the behavior is undefined.

# Core Animation Extensions To Key-Value Coding

The `CAAnimation` and `CALayer` classes extend the `NSKeyValueCoding` protocol adding default values for keys, expanded wrapping conventions, and key path support for `CGPoint`, `CGRect`, `CGSize`, and `CATransform3D`.

## Key-Value Coding Compliant Container Classes

Both `CALayer` and `CAAnimation` are key-value coding compliant container classes, allowing you to set values for arbitrary keys. That is, while the key “someKey” is not a declared property of the `CALayer` class, however you can still set a value for the key “someKey” as follows:

```
[theLayer setValue:[NSNumber numberWithInt:50] forKey:@"someKey"];
```

You retrieve the value for the key “someKey” using the following code:

```
someKeyValue=[theLayer valueForKey:@"someKey"];
```

**Mac OS X Note:** On Mac OS X, the `CALayer` and `CAAnimation` classes support the `NSCoding` protocol and will automatically archive any additional keys that you set for an instance of those classes.

## Default Value Support

Core Animation adds a new convention to key value coding that allows a class to provide a default value that is used when a class has no value set for that key. Both `CALayer` or `CAAnimation` support this convention using the class method `defaultValueForKey:`.

To provide a default value for a key you create a subclass of the class and override `defaultValueForKey:`. The subclass implementation examines the key parameter and then returns the appropriate default value. Listing 1 shows an example implementation of `defaultValueForKey:` that provides a new default value for the layer property `masksToBounds`.

**Listing 1** Example implementation of `defaultValueForKey:`

```
+ (id)defaultValueForKey:(NSString *)key
{
    if ([key isEqualToString:@"masksToBounds"])
        return [NSNumber numberWithBool:YES];

    return [super defaultValueForKey:key];
}
```

## Wrapping Conventions

When using the key-value coding methods to access properties whose values are not objects the standard key-value coding wrapping conventions support, the following wrapping conventions are used:

C Type	Class
CGPoint	NSNumber
CGSize	NSNumber
CGRect	NSNumber
CGAffineTransform	NSAffineTransform
CATransform3D	NSNumber

## Key Path Support for Structure Fields

`CAAnimation` provides support for accessing the fields of selected structures using key paths. This is useful for specifying these structure fields as the key paths for animations, as well as setting and getting values using `setValueForKeyPath:` and `valueForKeyPath:`.

`CATransform3D` exposes the following fields:

Structure Field	Description
<code>rotation.x</code>	The rotation, in radians, in the x axis.
<code>rotation.y</code>	The rotation, in radians, in the y axis.
<code>rotation.z</code>	The rotation, in radians, in the z axis.
<code>rotation</code>	The rotation, in radians, in the z axis. This is identical to setting the <code>rotation.z</code> field.
<code>scale.x</code>	Scale factor for the x axis.
<code>scale.y</code>	Scale factor for the y axis.
<code>scale.z</code>	Scale factor for the z axis.
<code>scale</code>	Average of all three scale factors.
<code>translation.x</code>	Translate in the x axis.
<code>translation.y</code>	Translate in the y axis.
<code>translation.z</code>	Translate in the z axis.
<code>translation</code>	Translate in the x and y axis. Value is an <code>NSSize</code> or <code>CGSize</code> .

`CGPoint` exposes the following fields:

Structure Field	Description
<code>x</code>	The x component of the point.
<code>y</code>	The y component of the point.

`CGSize` exposes the following fields:

Structure Field	Description
<code>width</code>	The width component of the size.
<code>height</code>	The height component of the size.

`CGRect` exposes the following fields:

Structure Field	Description
<code>origin</code>	The origin of the rectangle as a <code>CGPoint</code> .
<code>origin.x</code>	The x component of the rectangle origin.
<code>origin.y</code>	The y component of the rectangle origin.
<code>size</code>	The size of the rectangle as a <code>CGSize</code> .
<code>size.width</code>	The width component of the rectangle size.
<code>size.height</code>	The height component of the rectangle size.

You can not specify a structure field key path using Objective-C 2.0 properties. This will not work:

```
myLayer.transform.rotation.x=0;
```

Instead you must use `setValue:forKeyPath:` or `valueForKeyPath:` as shown below:

```
[myLayer setValue:[NSNumber numberWithInt:0]
 forKeyPath:@"transform.rotation.x"];
```



# Layer Style Properties

---

Regardless of the type of media a layer displays, a layer's style properties are applied by the render-tree as it composites layers.

This chapter describes the layer style properties and provides examples of their effect on an example layer.

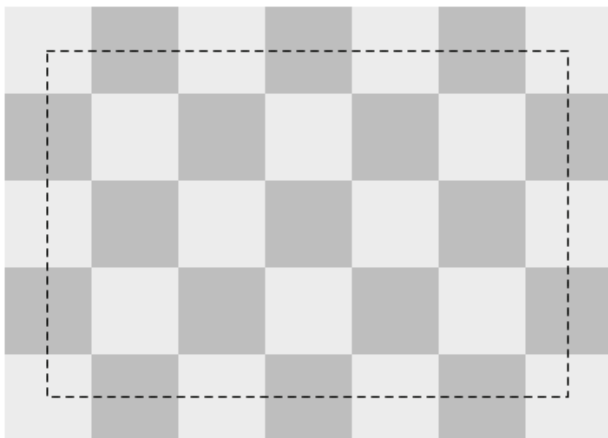
**Note:** The layer style properties available on Mac OS X and iOS differ and are noted below.

## Geometry Properties

A layer's geometry properties specify how it is displayed relative to its parent layer. The geometry also specifies the radius used to round the layer corners (available only on Mac OS X) and a transform that is applied to the layer and its sublayers.

Figure 1 shows the geometry of the example layer.

**Figure 1** Layer geometry



The following `CALayer` properties specify a layer's geometry:

- [frame](#) (page 96)
- [bounds](#) (page 91)
- [position](#) (page 101)
- [anchorPoint](#) (page 88)
- [cornerRadius](#) (page 94)

- [transform](#) (page 105)
- [zPosition](#)

**iOS Note:** Only iOS v 3.0 and later supports the `cornerRadius` property.

## Background Properties

Next, the layer renders its background. You can define a color for the background as well as a Core Image filter.

Figure 2 illustrates the sample layer with its `backgroundColor` set.

**Figure 2** Layer with background color



The background filter is applied to the content behind the layer. For example, you may wish to apply a blur filter as a background filter to make the layer content stand out better.

The following `CALayer` properties affect the display of a layer's background:

- [backgroundColor](#) (page 89)
- [backgroundFilters](#) (page 89)

**iOS Note:** While the `CALayer` class in iOS exposes the `backgroundFilters` property, Core Image is not available. The filters available for this property are currently undefined.

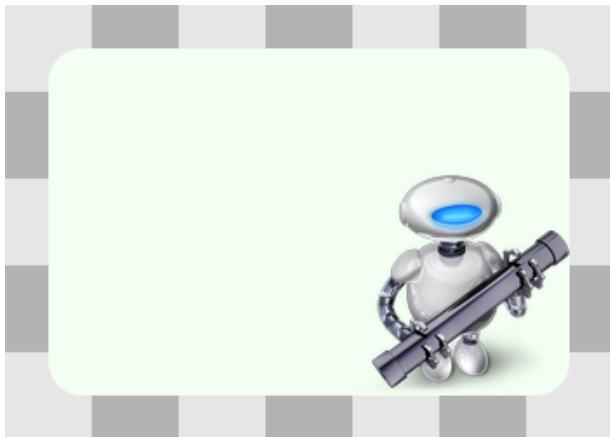
## Layer Content

Next, if set, the content of the layer is rendered. The layer content can be created using the Quartz graphics environment, OpenGL, QuickTime, or Quartz Composer.



Figure 4 shows the example layer with its content composited.

**Figure 3** Layer displaying a content image



By default, the content of a layer is not clipped to its bounds and corner radius. The `masksToBounds` property can be set to `true` to clip the layer content to those values.

The following `CALayer` properties affect the display of a layer's content:

- `contents` (page 92)
- `contentsGravity` (page 93)

## Sublayers Content

It is typical that a layer will have a hierarchy of child layers, its sublayers. These sublayers are rendered recursively, relative to the parent layer's geometry. The parent layer's `sublayerTransform` is applied to each sublayer, relative to the parent layer's anchor point.

**Figure 4** Layer displaying the sublayers content



By default, a layer's sublayers are not clipped to the layer's bounds and corner radius. The `maskToBounds` property can be set to `true` to clip the layer content to those values. The example layer's `maskToBounds` property is `false`; notice that the sublayer displaying the monitor and test pattern is partially outside of its parent layer's bounds.

The following `CALayer` properties affect the display of a layer's sublayers:

- `sublayers` (page 104)
- `maskToBounds` (page 99)
- `sublayerTransform` (page 104)

## Border Attributes

A layer can display an optional border using a specified color and width. Figure 5 shows the example layer after applying a border.

**Figure 5** Layer displaying the border attributes content



The following `CALayer` properties affect the display of a layer's borders:

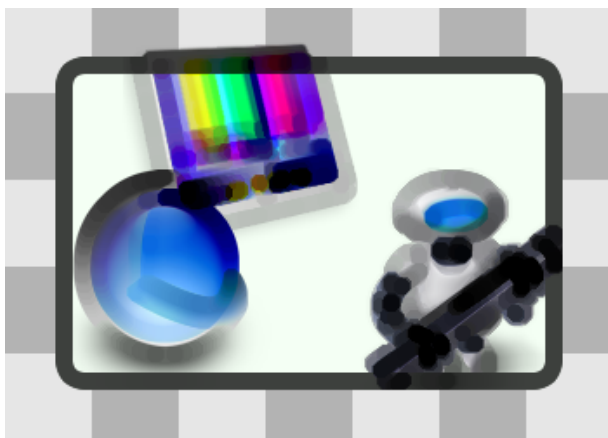
- `borderColor` (page 90)
- `borderWidth` (page 90)

**iOS Note:** As a performance consideration, iOS does not support the `borderColor` and `borderWidth` properties. Drawing a border for layer content is the responsibility of the developer.

## Filters Property

An array of Core Image filters can be applied to the layer. These filters affect the layer's border, content, and background. Figure 6 shows the example layer with the Core Image posterize filter applied.

**Figure 6** Layer displaying the filters properties



The following `CALayer` property specifies a layers content filters:

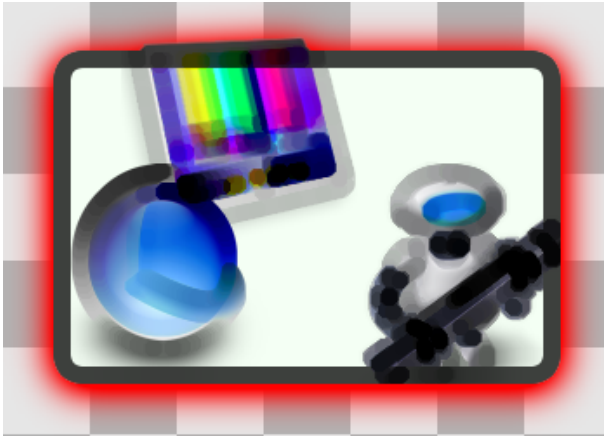
- `filters` (page 96)

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

## Shadow Properties

Optionally, a layer can display a shadow, specifying its opacity, color, offset, and blur radius. Figure 7 shows the example layer with a red shadow applied.

**Figure 7** Layer displaying the shadow properties



The following `CALayer` properties affect the display of a layer's shadow:

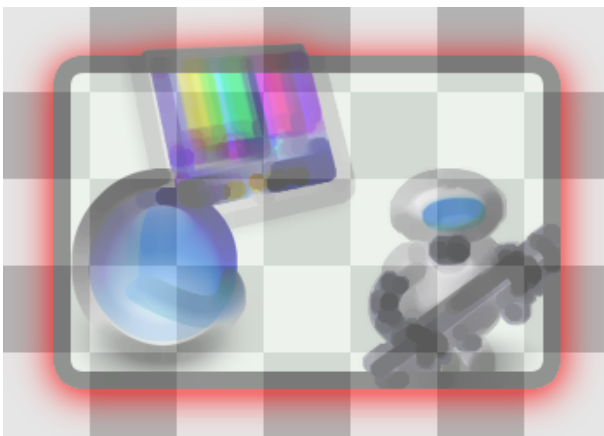
- `shadowColor` (page 102)
- `shadowOffset` (page 102)
- `shadowOpacity` (page 102)
- `shadowRadius` (page 103)

**iOS Note:** As a performance consideration, iOS does not support the `shadowColor`, `shadowOffset`, `shadowOpacity`, and `shadowRadius` properties.

## Opacity Property

By setting the opacity of a layer, you can control the layer's transparency. Figure 8 shows the example layer with an opacity of 0.5.

**Figure 8** Layer including the opacity property



The following `CALayer` property specifies the opacity of a layer:

- `opacity` (page 100)

## Composite Property

A layer's compositing filter is used to combine the layer content with the layers behind it. By default, a layer is composited using source-over. Figure 9 shows the example layer with a compositing filter applied.

**Figure 9** Layer composited using the `compositingFilter` property



The following `CALayer` property specifies the compositing filter for a layer:

- `compositingFilter` (page 91)

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

## Mask Properties

Finally, you can specify a layer that will serve as a mask, further modifying how the rendered layer appears. The opacity of the mask layer determines masking when the layer is composited. Figure 10 shows the example layer composited with a mask layer.

**Figure 10** Layer composited with the mask property



The following `CALayer` property specifies the mask for a layer:

- `mask` (page 98)

**iOS Note:** As a performance consideration, iOS does not support the `mask` property.

# Example: Core Animation Kiosk Menu Style Application

---

The Core Animation Kiosk Style Menu example displays a simple selection example using Core Animation layers to generate and animate the user interface. In less than 100 lines of code, it demonstrates the following capabilities and design patterns:

- Hosting the root-layer of a layer hierarchy in a view.
- Creating and inserting layers into a layer hierarchy.
- Using a `QCCompositionLayer` to display Quartz Composer compositions as layer content. It also demonstrates the performance increase when using a solid color.
- Using an explicit animation that runs continuously.
- Animating Core Image Filter inputs.
- Implicitly animating the position of the selection item.
- Handling key events through the `MenuView` instance that hosts the view.

This application makes heavy use of Core Image filters and Quartz Composer compositions and, as a result, runs only on Mac OS X. The techniques illustrated for managing the layer hierarchy, implicit and explicit animation, and event handling are common to both platforms.

The application is available as sample code in two versions: One that uses the `QCCompositionLayer` as the background and a second that uses a solid black background. See the *CoreAnimationKioskStyleMenu* sample for a working copy of the application. The initial code walkthrough describes the version of the application that uses the Quartz Composer background. The performance increase when using a solid color is discussed later in the article.

## The User Interface

The `QCCoreAnimationKioskMenuStyle` sample provides a very basic kiosk-style user interface; it takes over the entire screen and the user can select a single item in a menu. The user navigates the menu using the up and down arrows on the keyboard. As the selection changes the selection indicator (the rounded white rectangle) animates to its new location. A continuously animating bloom filter is set for the selection indicator causing it to subtly catch your attention. The background is a Quartz Composer animation runs continuously, providing an attention grabbing background. Figure 1 shows the application's interface.

**Figure 1** Core Animation Kiosk Menu Interface



## Examining the Nib File

---

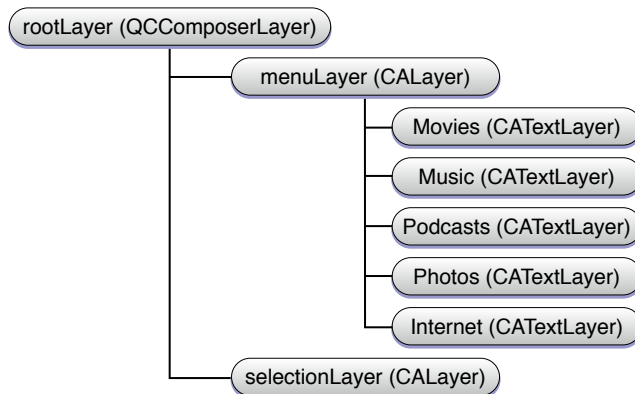
### The Layer Hierarchy

---

The layer hierarchy, also referred to as the layer tree, of the CoreAnimationKioskMenuStyle application is shown below.



**Figure 2** Layer Hierarchy For QCCoreAnimationKioskStyleMenu Application



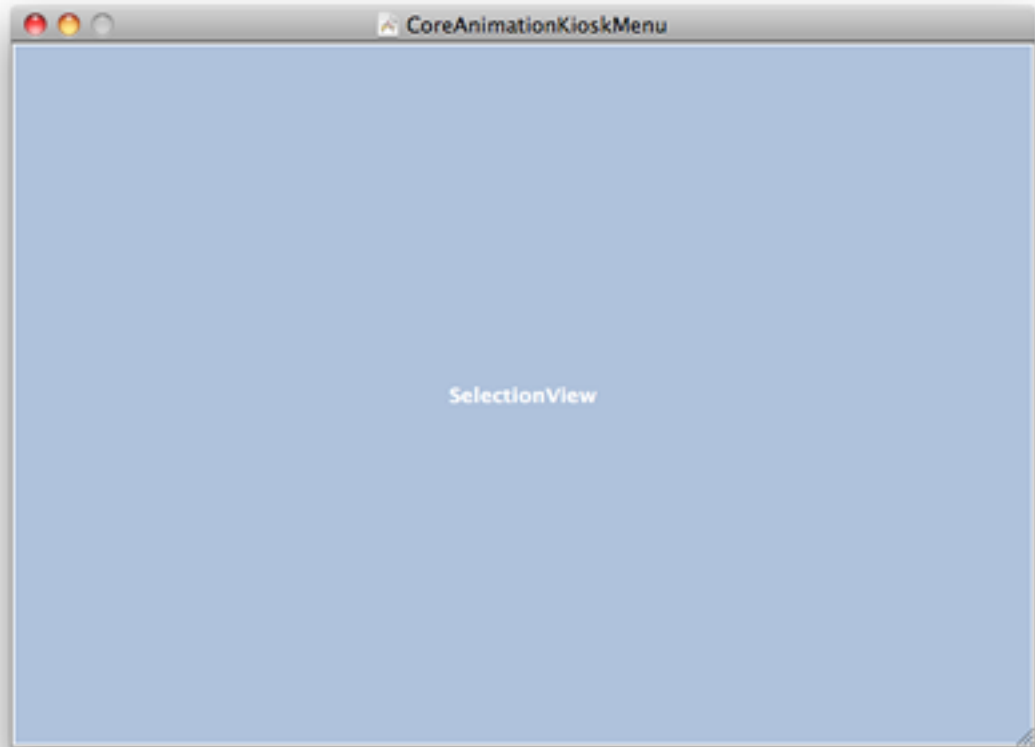
The `rootLayer` is an instance of `QCComposerLayer`. As the root-layer this layer is the same size as the `MenuView` instance, and remains that way as the window is resized.

The `menuLayer` is a sublayer of the `rootLayer`. It is an empty layer; it does not have anything set as its `contents` property and none of its style properties are set. The `menuLayer` is simply used as a container for the menu item layers. This approach allows the application to easily access a menu item sublayer by its position in the `menuLayers.sublayers` array. The `menuLayer` is the same size as, and overlaps, the `rootLayer`. This was done intentionally so that there was no need to convert between coordinate systems when positioning the `selectionLayer` relative to the current menu item.

## Examining the Application's Nib

`MainMenu.nib` is very straightforward. An instance of `CustomView` is dragged from the Interface Builder palette and positioned in the window. It is resized such that it fills the entire window. The `SelectionView.h` file is imported into Interface Builder by dragging it to the `Menu.nib` window. The `CustomView` is then selected in the Selection View Identity palette, and the Class is changed to `SelectionView`.

Figure 3 MainMenu.nib File



No other connections need to be made. When the nib file is loaded the window is unarchived along with the `SelectionView` instance. The `SelectionView` class layers are configured in the class's implementation of `awakeFromNib`.

## Examining the Application's Code

Having looked at the application's nib file and the overall design, you can now begin examining the implementation of the `SelectionView` class. The `SelectionView.h` file declares the properties and methods for the `SelectionView`, `SelectionView.m` contains the implementation. In order to find the code snippets easier in the implementation file, `#pragma mark` statements have been added that correspond to each listing.

## QCCoreAnimationKioskStyleMenu.h and QCCoreAnimationKioskStyleMenu.m Files

---

The CodeAnimationKioskMenuAppDelegate.h and .m files were created automatically when the application was created. They contain no code of relevance to this example.

### Examining SelectionView.h

---

The `SelectionView` class is a subclass of `NSView`. It declares four properties as well as the methods the `SelectionView.h` implements. It is the only view in the application's window and serves as the container for the `rootLayer` and the other layers in the applications layer tree.

#### Listing 1 SelectionView.h File Listing

```
#import <Cocoa/Cocoa.h>
#import <QuartzCore/Quartz.h>

// The SelectionView class is the view subclass that is inserted into
// the window. It hosts the rootLayer, and responds to events.
@interface SelectionView : NSView {

    // Contains the selected menu item index
    NSInteger selectedIndex;

    // The layer that contains the menu item layers
    CALayer *menuLayer;

    // The layer that is displays the selection
    CALayer *selectionLayer;

    // The array of layers that contain the menu item names.
    NSArray *names;

}

@property NSInteger selectedIndex;
@property (retain) CALayer *menuLayer;
@property (retain) CALayer *selectionLayer;
@property (retain) NSArray *names;

-(void)awakeFromNib;
-(void)setupLayers;
-(void)changeSelectedIndex:(NSInteger)theSelectedIndex;
-(void)moveUp:(id)sender;
-(void)moveDown:(id)sender;
-(void)dealloc;

@end
```

**Note:** Notice that `Quartz/CoreAnimation.h` is imported. The `QuartzCore.framework` must be added to any project that uses Core Animation. Because this example uses Quartz Composer the `SelectionView.h` header also imports `Quartz/Quartz.h`, and the `Quartz.framework` is added to the project.

## Examining SelectionView.m

---

The `SelectionView` class is the workhorse of this application. It responds when the view is loaded by the nib, expands to fill the current screen, sets up the layers to be displayed, creates the animations, and handles the keys that move the selection.

The listing of the `SelectionView.m` file is divided as follows:

- The implementation of `awakeFromNib`.
- Setting Up the Layers
- Animating the Selection Layer Movement
- Responding to Key Events
- Cleaning Up

### Implementing `awakeFromNib`

---

The `awakeFromNib` method is called when `MainMenu.nib` is loaded and unarchived. The view is expected to complete its setup in `awakeFromNib`.

The `MainMenuView` implementation of `awakeFromNib` does the following:

- Creates an array of strings, the names, which are used to display the menu items.
- Hides the cursor. Full screen applications typically don't show the cursor and this application relies entirely on keyboard input.
- The view is resized to automatically fill the screen in which the window resizes. This gives the kiosk look.
- Makes the window the view resides within the first responder so that it receives the up and down arrow events.
- Calls the `setupLayers` method to setup the layers (discussed in “[The setupLayers Method](#)” (page 301))
- Finally, the window is brought to the front and made visible.

The code excerpt in Listing 2 shows the code that is used to accomplish the above functionality. It is located at the `#pragma mark "Implementation of awakeFromNib"` in `SelectionView.m` within the example project.

#### Listing 2 Implementation of `awakeFromNib`

```
#pragma mark - Listing 1: Implementation of awakeFromNib
- (void)awakeFromNib
{
    // create an array that contains the various
    // strings
```

```

self.names=[NSArray arrayWithObjects:@"Movies",@"Music",
    @"Podcasts",@"Photos",@"Internet",
    nil] ;

// The cursor isn't used for selection, so we hide it
[NSCursor hide];

// go full screen, as a kiosk application
[self enterFullScreenMode:[self.window screen] withOptions:NULL];

// Make the window the first responder to get keystrokes
[self.window makeFirstResponder:self];

// setup the individual layers
[self setupLayers];

// bring the window to the front
[self.window makeKeyAndOrderFront:self];
}

```

## The setupLayers Method

---

The majority of the code in the `QCCoreAnimationKioskStyleMenu` application is in the `setupLayers` method. This method is responsible for creating the layers and animations and configuring the currently selected item.

The `#pragma` mark "Configuration of the Background rootLayer" is shown in [Listing 3](#) (page 301). This snippet of code sets up the `rootLayer` for all the child layers.

There are two significant element to take notice of in this section of code:

- The `rootLayer` is created and set for the view and only then layers are enabled for the view. Doing this in this order causes the view to use the specified layer, rather than creating it's own layer, this is referred to as **layer-view hosting**. Layer-view hosting requires that all drawing be the responsibility of Core Animation. You must not use the `NSView` drawing capabilities and methods, however the view event-handling routines are used as normal.
- The other is related to performance. Depending upon the hardware, running a Quartz Composer animation in the background can cause performance issues. Using a solid color performance can be increased. This is further discussed in "[Performance Considerations](#)" (page 305).

### Listing 3 Configuration of the Background rootLayer

```

#pragma mark Listing: "Configuration of the Background rootLayer"

-(void)setupLayers;
{
    // make a Quartz Composition Layer.
    // Note: Running a QCComposition can significantly impact performance
    QCCompositionLayer* rootLayer=[QCCompositionLayer compositionLayerWithFile:
        [[NSBundle mainBundle] pathForResource:@"Background"
ofType:@"qtz"]];

    // Set the QCCompositionLayer as the root layer
    // and then turn on wantsLayer. This order cases

```

```
// layer-hosting behavior on the part of the view..
[self setLayer:rootLayer];
[self setWantsLayer:YES];
```

The `#pragma mark Setup menuLayers Array. The Selectable Menu Items.` is shown in [Listing 4](#) (page 302). This code initialized the `menuLayer` of the application inserting it as a sublayer of the `rootLayer` (see [Figure 2](#) (page 297) for a reminder of the layer hierarchy of the application. It then inserts separate layers for each text entry into the `menuLayer`. This code is located at `#pragma mark - Setup menuLayers Array. The Selectable Menu Items.` in the sample.

**Listing 4** Setup menuLayers Array. The Selectable Menu Items.

```
// Create a layer to contain the menus
self.menuLayer=[CALayer layer] ;
self.menuLayer.frame=rootLayer.frame;
self.menuLayer.layoutManager=[CAConstraintLayoutManager layoutManager];
[rootLayer addSublayer:self.menuLayer];

// setup and calculate the size and location of the individually selectable
items.
CGFloat width=400.0;
CGFloat height=50.0;
CGFloat spacing=20.0;
CGFloat fontSize=32.0;
CGFloat initialOffset=self.bounds.size.height/2-(height*5+spacing*4)/2.0;

//Create whiteColor it's used to draw the text and also in the selectionLayer
CGColorRef whiteColor=CGColorCreateGenericRGB(1.0f,1.0f,1.0f,1.0f);

// Iterate over the list of selection names and create layers for each.
// The menuItemLayer's are also positioned during this loop.
NSInteger i;
for (i=0;i<[names count];i++) {

    CATextLayer *menuItemLayer=[CATextLayer layer];
    menuItemLayer.string=[self.names objectAtIndex:i];
    menuItemLayer.font=@"Lucida-Grande";
    menuItemLayer.fontSize=fontSize;
    menuItemLayer.foregroundColor=whiteColor;
    [menuItemLayer addConstraint:[CAConstraint
        constraintWithAttribute:kCAConstraintMaxY
        relativeTo:@"superlayer"
        attribute:kCAConstraintMaxY
        offset:-(i*height+spacing+initialOffset)]];
    [menuItemLayer addConstraint:[CAConstraint
        constraintWithAttribute:kCAConstraintMidX
        relativeTo:@"superlayer"
        attribute:kCAConstraintMidX]];
    [self.menuLayer addSublayer:menuItemLayer];
} // end of for loop
[self.menuLayer layoutIfNeeded];
```

The `#pragma mark Setup selectionLayer. Used to Display the Currently Selected Item.` is shown in [Listing 5](#) (page 303). The code is commented, but to summarize:

- The `selectionLayer` `CALayer` is created.
- A `CIBloom` filter is added to the layer and configured to continuously provide a pulsing animation to the selection indicator.
- The `selectedIndex` is set to 0, the initial value.
- The `selectionLayer` is added to the `rootLayer` as a sublayer.

The implementation can be found at the `#pragma mark - Setup selectionLayer`. Displays the Currently Selected Item. in the sample code

**Listing 5**      `Setup selectionLayer`. Used to Display the Currently Selected Item.

```
#pragma mark - Setup selectionLayer. Displays the Currently Selected Item.

// we use an additional layer, selectionLayer
// to indicate that the current item is selected
self.selectionLayer=[CALayer layer];
self.selectionLayer.bounds=CGRectMake(0.0,0.0,width,height);
self.selectionLayer.borderWidth=2.0;
self.selectionLayer.cornerRadius=25;
self.selectionLayer.borderColor=whiteColor;
CIFilter *filter = [CIFilter filterWithName:@"CIBloom"];
[filter setDefaults];
[filter setValue:[NSNumber numberWithFloat:5.0] forKey:@"inputRadius"];
[filter setName:@"pulseFilter"];
[selectionLayer setFilters:[NSArray arrayWithObject:filter]];

// The selectionLayer shows a subtle pulse as it
// is displayed. This section of the code create the pulse animation
// setting the filters.pulsefilter.inputIntensity to range from 0 to 2.
// This will happen every second, autoreverse, and repeat forever
CABasicAnimation* pulseAnimation = [CABasicAnimation animation];
pulseAnimation.keyPath = @"filters.pulseFilter.inputIntensity";
pulseAnimation.fromValue = [NSNumber numberWithFloat: 0.0];
pulseAnimation.toValue = [NSNumber numberWithFloat: 2.0];
pulseAnimation.duration = 1.0;
pulseAnimation.repeatCount = 1e100f;
pulseAnimation.autoreverses = YES;
pulseAnimation.timingFunction = [CAMediaTimingFunction functionWithName:
                               kCAMediaTimingFunctionEaseInEaseOut];
[selectionLayer addAnimation:pulseAnimation forKey:@"pulseAnimation"];

// set the first item as selected
[self changeSelectedIndex:0];

// finally, the selection layer is added to the root layer
[rootLayer addSublayer:self.selectionLayer];

// cleanup
CGColorRelease(whiteColor);
// end of setupLayers
}
```

## Responding to Key Events

---

Because layers do not take part in the responder chain, or accept events, the `SelectionView` view instance that acts as the **layer-host** for the layer tree must assume that role. This is why it was registered as the first responder in the `awakeFromNib` method.

The `moveUp:` and `moveDown:` messages are provided by `NSResponder`, of which the `SelectionView` class is a descendent, as are all view classes. The `moveUp:` and `moveDown:` messages are invoked when the up arrow and down arrows are pressed respectively. These methods allows the application to respect any remapped arrow key functionally specified by the user. And it's easier than implementing `keyDown:`, although more complex key handling in your own application may require you do to so.

The method implementation shown in [Listing 6](#) (page 304) implements the setting of the current selection layer. You can find it in the sample code at `#pragma mark "Handle Changes in the Selection"`.

### Listing 6 Handle Changes in Selection

```
#pragma mark Handle Changes in the Selection

-(void)changeSelectedIndex:(NSInteger)theSelectedIndex
{
    self.selectedIndex=theSelectedIndex;

    if (self.selectedIndex == [names count])
        self.selectedIndex=[names count]-1;
    if (self.selectedIndex < 0)
        self.selectedIndex=0;

    CALayer *theSelectedLayer=[[self.menuLayer sublayers]
objectAtIndex:self.selectedIndex];

    // Moves the selectionLayer to illustrate the
    // currently selected item. It does this
    // using an animation so that the transition
    // is visible.
    self.selectionLayer.position=theSelectedLayer.position;
};
```

When the up arrow is pressed the `selectedIndex` value is decremented and updated by calling `changeSelectedIndex:`. As shown in [Listing 6](#) (page 304) this moves the selection to highlight the correct item. The sample code section is shown at `#pragma mark Handle Keystrokes`.

### Listing 7 Handling Up and Down Key Presses

```
#pragma mark Handle Keystrokes

-(void)moveUp:(id)sender
{
    [self changeSelectedIndex:self.selectedIndex-1];
}

-(void)moveDown:(id)sender
{
    [self changeSelectedIndex:self.selectedIndex+1];
}
```



```
}
```

When the `SelectionView` is closed, we are responsible for cleaning up our instance variables. The `menuLayer`, `selectionLayer`, and `names` are set to `nil` (which results in their release) in the `dealloc` implementation. The `#pragma mark Dealloc and Cleanup` shows this portion of the sample code.

#### Listing 8 Dealloc and Cleanup

```
#pragma mark Dealloc and Cleanup
```

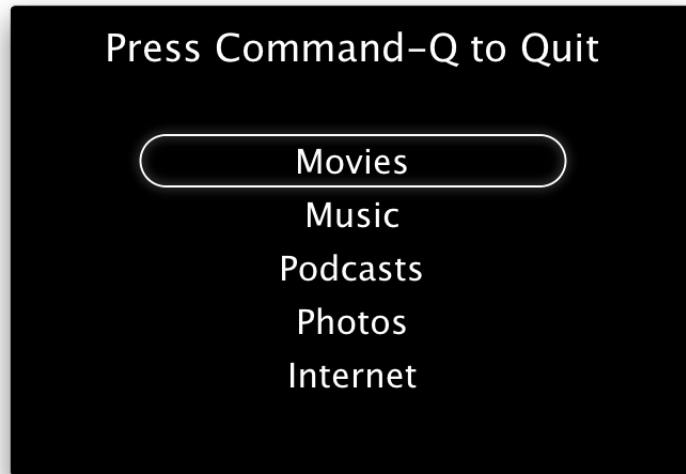
```
-(void)dealloc  
{  
    [self setLayer:nil];  
    self.menuLayer=nil;  
    self.selectionLayer=nil;  
    self.names=nil;  
    [super dealloc];  
}
```

## Performance Considerations

Setting a Quartz Composer animation as the background for this type of interface makes for a much more interesting and eye catching display. However, you should be sure to do the required performance testing on the target hardware configurations. Not all hardware configurations can maintain smooth animation of the selection indicator when it runs on top of a Quartz Composer animation. You may want to consider using a solid color or a static image for the background.

To help you evaluate any performance impact the sample code project includes an additional project **CoreAnimationKioskStyleMenu** that draws the content over a solid background rather than a Quartz Composer animation. It may be useful to compare the performance of the two applications as an example of the potential impact.

**Figure 4** Alternate Interface With Black Background.



# Animatable Properties

---

Many of the properties in `CALayer` and `CIFilter` can be animated. This article lists those properties, along with the animation used by default.

## CALayer Animatable Properties

The following `CALayer` class properties can be animated by Core Animation. See `CALayer` for more information.

- [anchorPoint](#) (page 88)  
Uses the default implied `CABasicAnimation` described in [Table 1](#) (page 309).
- [backgroundColor](#) (page 89)  
Uses the default implied `CABasicAnimation` described in [Table 1](#) (page 309). (subproperties are animated using a basic animation)
- [backgroundFilters](#) (page 89)  
Uses the default implied `CATransitionAnimation` described in [Table 2](#) (page 309). Sub-properties of the filters are animated using the default implied `CABasicAnimation` described in [Table 1](#) (page 309).
- [borderColor](#) (page 90)  
Uses the default implied `CABasicAnimation` described in [Table 1](#) (page 309).
- [borderWidth](#) (page 90)  
Uses the default implied `CABasicAnimation` described in [Table 1](#) (page 309).
- [bounds](#) (page 91)  
Uses the default implied `CABasicAnimation` described in [Table 1](#) (page 309).
- [compositingFilter](#) (page 91)  
Uses the default implied `CATransitionAnimation` described in [Table 2](#) (page 309). Sub-properties of the filters are animated using the default implied `CABasicAnimation` described in [Table 1](#) (page 309).
- [contents](#) (page 92)
- [contentsRect](#) (page 94)  
Uses the default implied `CABasicAnimation` described in [Table 1](#) (page 309).
- [cornerRadius](#) (page 94)  
Uses the default implied `CABasicAnimation` described in [Table 1](#) (page 309).
- [doubleSided](#) (page 95)  
No default implied animation is set.

- [filters](#) (page 96)  
Uses the default implied `CABasicAnimation` described in [Table 1](#) (page 309). Sub-properties of the filters are animated using the default implied `CABasicAnimation` described in [Table 1](#) (page 309).
- [frame](#) (page 96)  
The frame property itself is not animatable. You can achieve the same results by modifying the `bounds` and `position` properties instead.
- [hidden](#) (page 97)  
Uses the default implied `CABasicAnimation` described in [Table 1](#) (page 309).
- [mask](#) (page 98)  
Uses the default implied `CABasicAnimation` described in [Table 1](#) (page 309). This property is available only on Mac OS X.
- [masksToBounds](#) (page 99)  
Uses the default implied `CABasicAnimation` described in [Table 1](#) (page 309).
- [opacity](#) (page 100)  
Uses the default implied `CABasicAnimation` described in [Table 1](#) (page 309).
- [position](#) (page 101)  
Uses the default implied `CABasicAnimation` described in [Table 1](#) (page 309).
- [shadowColor](#) (page 102)  
Uses the default implied `CABasicAnimation` described in [Table 1](#) (page 309). This property is available only on Mac OS X.
- [shadowOffset](#) (page 102)  
Uses the default implied `CABasicAnimation` described in [Table 1](#) (page 309). This property is available only on Mac OS X.
- [shadowOpacity](#) (page 102)  
Uses the default implied `CABasicAnimation` described in [Table 1](#) (page 309). This property is available only on Mac OS X.
- [shadowRadius](#) (page 103)  
Uses the default implied `CABasicAnimation` described in [Table 1](#) (page 309). This property is available only on Mac OS X.
- [sublayers](#) (page 104)  
Uses the default implied `CATransitionAnimation` described in [Table 2](#) (page 309).
- [sublayerTransform](#) (page 104)  
Uses the default implied `CABasicAnimation` described in [Table 1](#) (page 309).
- [transform](#) (page 105)  
Uses the default implied `CABasicAnimation` described in [Table 1](#) (page 309).
- [zPosition](#) (page 106)  
Uses the default implied `CABasicAnimation` described in [Table 1](#) (page 309).

**Table 1** Default Implied Basic Animation

Description	Value
Class	CABasicAnimation
duration	.25 seconds, or the duration of the current transaction
keyPath	Dependent on layer property type

**Table 2** Default Implied Transition

Description	Value
Class	CATransition
duration	.25 seconds, or the duration of the current transaction
type	Fade (kCATransitionFade)
startProgress	0.0
endProgress	1.0

## CIFilter Animatable Properties

Core Animation adds the following animatable properties to Core Image’s `CIFilter` class. See `CIFilter` [Core Animation Additions](#) for more information. These properties are available only on Mac OS X.

- [name](#) (page 208)
- [enabled](#) (page 208)



# Document Revision History

---

This table describes the changes to *Core Animation Reference Collection*.

Date	Notes
2009-07-15	Added CADisplayLink
2009-03-10	Updated for iOS 3.0.
2008-06-26	Updated for iOS.
2007-10-31	Reordered companion guides to indicate prerequisites.
2007-07-24	New document that describes the Objective-C API for Core Animation .

**REVISION HISTORY**

Document Revision History