Quartz Core Framework Reference

Graphics & Imaging > Quartz

2008-03-12

Contents

Introduction

Framework	/System/Library/Frameworks/QuartzCore.framework
Header file directories	/System/Library/Frameworks/QuartzCore.framework/Headers
Declared in	CAAnimation.h CABase.h CACIFilterAdditions.h CAConstraintLayoutManager.h CALayer.h CAMediaTiming.h CAMediaTimingFunction.h CAOpenGLLayer.h CARenderer.h CAScrollLayer.h CATextLayer.h CATiledLayer.h CATransaction.h CATransform3D.h CIColor.h CIContext.h CIFilter.h CIFilterGenerator.h CIFilterShape.h CIImage.h CIImageAccumulator.h CIImageProvider.h CIKernel.h CIPlugIn.h CIPlugInInterface.h CIRAWFilter.h CISampler.h CIVector.h CVBase.h CVBuffer.h CVDisplayLink.h CVHostTime.h CVImageBuffer.h CVOpenGLBuffer.h CVOpenGLBufferPool.h CVOpenGLTexture.h CVOpenGLTextureCache.h CVPixelBuffer.h CVPixelBufferPool.h CVPixelFormatDescription.h CVReturn.h
Companion guides	Core Image Programming Guide

Image Unit Tutorial
Core Image Kernel Language Reference
Core Image Filter Reference
Core Video Programming Guide

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

Classes

CAAnimation Class Reference

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

Overview

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

Tasks

Archiving Properties

`– shouldArchiveValueForKey:` (page 19)
Specifies whether the value of the property for a given key is archived.

Providing Default Values for Properties

`+ defaultValueForKey:` (page 18)
Specifies the default value of the property with the specified key.

Creating an Animation

`+ animation` (page 17)
Creates and returns a new `CAAnimation` instance.

Animation Attributes

`removedOnCompletion` (page 17) *property*
Determines if the animation is removed from the target layer's animations upon completion.

`- isRemovedOnCompletion` (page 18)
A synthesized accessor for the `removedOnCompletion` (page 17) property.

`timingFunction` (page 17) *property*
An optional timing function defining the pacing of the animation.

Getting and Setting the Delegate

`delegate` (page 16) *property*
Specifies the receiver's delegate object.

Animation Progress

`- animationDidStart:` (page 19) *delegate method*
Called when the animation begins its active duration.

`- animationDidStop:finished:` (page 19) *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 2.0 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 for Cocoa*.

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.

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.

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.

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 17) property.

```
- (BOOL)isRemovedOnCompletion
```

See Also
`@property removedOnCompletion` (page 17)

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

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

Overview

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

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

Important: The `delegate` and `removedOnCompletion` properties of animations in the `animations` (page 22) array are currently ignored. The `CAAnimationGroup` delegate does receive these messages.

Note: The `delegate` and `removedOnCompletion` properties of animations in the `animations` (page 22) property are currently ignored.

Tasks

Grouped Animations

`animations` (page 22) *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 2.0 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

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

Overview

`CABasicAnimation` provides basic, single-keyframe animation capabilities for a layer property. You create an instance of `CABasicAnimation` using the inherited `animationWithKeyPath:` (page 103) method, specifying the key path of the property to be animated in the render tree.

Setting Interpolation Values

The `fromValue` (page 24), `byValue` (page 24) and `toValue` (page 25) 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 24) and `toValue` (page 25) are non-`nil`. Interpolates between `fromValue` (page 24) and `toValue` (page 25).
- `fromValue` (page 24) and `byValue` (page 24) are non-`nil`. Interpolates between `fromValue` (page 24) and (`fromValue` (page 24) + `byValue` (page 24)).
- `byValue` (page 24) and `toValue` (page 25) are non-`nil`. Interpolates between (`toValue` (page 25) - `byValue` (page 24)) and `toValue` (page 25).
- `fromValue` (page 24) is non-`nil`. Interpolates between `fromValue` (page 24) and the current presentation value of the property.

- `toValue` (page 25) is non-`nil`. Interpolates between the current value of `keyPath` in the target layer's presentation layer and `toValue` (page 25).
- `byValue` (page 24) is non-`nil`. Interpolates between the current value of `keyPath` in the target layer's presentation layer and that value plus `byValue` (page 24).
- 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 24) *property*
Defines the value the receiver uses to start interpolation.

`toValue` (page 25) *property*
Defines the value the receiver uses to end interpolation.

`byValue` (page 24) *property*
Defines the value the receiver uses to perform relative interpolation.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

byValue

Defines the value the receiver uses to perform relative interpolation.

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

Discussion
See "Setting Interpolation Values" (page 23) 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 23) for details on how `fromValue` interacts with the other interpolation values.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CAAnimation.h`

toValue

Defines the value the receiver uses to end interpolation.

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

Discussion

See "Setting Interpolation Values" (page 23) for details on how `toValue` interacts with the other interpolation values.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CAAnimation.h`

CAConstraint Class Reference

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

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 29)
Creates and returns an `CAConstraint` object with the specified parameters.

`+ constraintWithAttribute:relativeTo:attribute:offset:` (page 29)
Creates and returns an `CAConstraint` object with the specified parameters.

`+ constraintWithAttribute:relativeTo:attribute:` (page 28)
Creates and returns an `CAConstraint` object with the specified parameters.

`- initWithAttribute:relativeTo:attribute:scale:offset:` (page 30)
Returns an `CAConstraint` object with the specified parameters. Designated initializer.

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.

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.

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

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

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 64) 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 34)
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

Declared In
`CAConstraintLayoutManager.h`

CAKeyframeAnimation Class Reference

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

Overview

`CAKeyframeAnimation` provides generic keyframe animation capabilities for a layer property in the render tree. You create an `CAKeyframeAnimation` instance using the inherited `animationWithKeyPath:` (page 103) 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 37) *property*
An optional `CGPathRef` that provides the keyframe values for the receiver.

`values` (page 38) *property*
An array of objects that provide the keyframe values for the receiver.

Keyframe Timing

`keyTimes` (page 36) *property*
An optional array of `NSNumber` objects that define the duration of each keyframe segment.

`timingFunctions` (page 38) *property*
An optional array of `CAMediaTimingFunction` instances that defines the pacing of the each keyframe segment.

`calculationMode` (page 36) *property*
Specifies how intermediate keyframe values are calculated by the receiver.

Rotation Mode

`rotationMode` (page 37) *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 2.0 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 39). The default is `kCAAnimationLinear` (page 39).

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 36) 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 keytimes.
- If the calculationMode is set to `kCAAnimationDiscrete`, the first value in the array must be 0.0.
- If the calculationMode is set to `kCAAnimationPaced`, 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 38) property. Each point in the path, except for moveto points, defines a single keyframe segment for the purpose of timing and interpolation. For constant velocity animation along the path, `calculationMode` (page 36) should be set to `kCAAnimationPaced` (page 39).

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

See Also
`@property rotationMode` (page 37)

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 38). 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 37)

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 37) 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 37) 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`.

Declared In

`CAAnimation.h`

Value calculation modes

These constants are used by the `calculationMode` (page 36) 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. This mode is not currently implemented

Available in Mac OS X v10.5 and later.

Declared in `CAAnimation.h`.

Declared In

`CAAnimation.h`

CALayer Class Reference

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

Overview

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

Tasks

Creating a Layer

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

`- init` (page 70)
  Returns an initialized `CALayer` object.

`- initWithLayer:` (page 70)
  Override to copy or initialize custom fields of the specified layer.

Accessing the Presentation Layer

`- presentationLayer` (page 74)
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 73)
Returns the model layer of the receiver, if it represents a current presentation layer.

Modifying the Layer Geometry

`frame` (page 54) *property*
Specifies receiver's frame rectangle in the super-layer's coordinate space.

`bounds` (page 50) *property*
Specifies the bounds rectangle of the receiver. Animatable.

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

`zPosition` (page 61) *property*
Specifies the receiver's position on the z axis. Animatable.

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

`- affineTransform` (page 65)
Convenience method for getting the `transform` (page 61) property as an affine transform.

`- setAffineTransform:` (page 78)
Convenience method for setting the `transform` (page 61) property as an affine transform.

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

`sublayerTransform` (page 60) *property*
Specifies a transform applied to each sublayer when rendering. Animatable.

Providing Layer Content

`contents` (page 51) *property*
An object that provides the contents of the layer. Animatable.

`contentsRect` (page 52) *property*
A rectangle, in the unit coordinate space, defining the subrectangle of `contents` (page 51) that the receiver should draw. Animatable.

`- display` (page 69)
Reload the content of this layer.

`- displayLayer:` (page 80) *delegate method*
Allows the delegate to override the `display` (page 69) implementation.

`- drawInContext:` (page 69)
Draws the receiver's content in the specified graphics context.

`- drawLayer:inContext:` (page 80) *delegate method*
Allows the delegate to override the layer's `drawInContext:` implementation.

`opaque` (page 57) *property*
  Specifies a hint marking that the pixel data provided by the `contents` (page 51) property is completely opaque.

– `isOpaque` (page 73)
  A synthesized accessor for the `opaque` (page 57) property.

`edgeAntialiasingMask` (page 53) *property*
  A bitmask defining how the edges of the receiver are rasterized.

`minificationFilter` (page 56) *property*
  The filter used when reducing the size of the content.

`magnificationFilter` (page 55) *property*
  The filter used when increasing the size of the content.

Style Attributes

`contentsGravity` (page 51) *property*
  Determines how the receiver's contents are positioned within its bounds.

`opacity` (page 57) *property*
  Determines the opacity of the receiver. Animatable.

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

– `isHidden` (page 72)
  A synthesized accessor for the `hidden` (page 54) property.

`masksToBounds` (page 56) *property*
  Determines if the sublayers are clipped to the receiver's bounds. Animatable.

`doubleSided` (page 53) *property*
  Determines whether the receiver is displayed when facing away from the viewer. Animatable.

– `isDoubleSided` (page 72)
  A synthesized accessor for the `doubleSided` (page 53) property.

`mask` (page 55) *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 52) *property*
  Specifies a radius used to draw the rounded corners of the receiver's background. Animatable.

`borderWidth` (page 49) *property*
  Specifies the width of the receiver's border. Animatable.

`borderColor` (page 49) *property*
  The color of the receiver's border. Animatable.

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

`backgroundFilters` (page 48) *property*
  An optional array of CoreImage filters that are applied to the receiver's background. Animatable.

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

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

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

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

`filters` (page 53) *property*
An array of CoreImage filters that are applied to the contents of the receiver and its sublayers. Animatable.

`compositingFilter` (page 50) *property*
A CoreImage filter used to composite the receiver's contents with the background. Animatable.

`style` (page 59) *property*
An optional dictionary referenced to find property values that aren't explicitly defined by the receiver.

Managing the Layer Hierarchy

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

`superlayer` (page 60) *property*
Specifies receiver's superlayer. (read-only)

`– addSublayer:` (page 65)
Appends the layer to the receiver's `sublayers` (page 60) array.

`– removeFromSuperlayer` (page 75)
Removes the layer from the `sublayers` (page 60) array or `mask` (page 55) property of the receiver's `superlayer` (page 60).

`– insertSublayer:atIndex:` (page 71)
Inserts the layer as a sublayer of the receiver at the specified index.

`– insertSublayer:below:` (page 72)
Inserts the layer into the receiver's sublayers array, below the specified sublayer.

`– insertSublayer:above:` (page 71)
Inserts the layer into the receiver's sublayers array, above the specified sublayer.

`– replaceSublayer:with:` (page 76)
Replaces the layer in the receiver's sublayers array with the specified new layer.

Updating Layer Display

`– setNeedsDisplay` (page 78)
Marks the receiver as needing display before the content is next committed.

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

`– setNeedsDisplayInRect:` (page 78)
Marks the region of the receiver within the specified rectangle as needing display.

Layer Animations

`- addAnimation:forKey:` (page 64)
: Add an animation object to the receiver's render tree for the specified key.

`- animationForKey:` (page 65)
: Returns the animation added to the receiver with the specified identifier.

`- removeAllAnimations` (page 74)
: Remove all animations attached to the receiver.

`- removeAnimationForKey:` (page 75)
: Remove the animation attached to the receiver with the specified key.

Managing Layer Resizing and Layout

`layoutManager` (page 55) *property*
: Specifies the layout manager responsible for laying out the receiver's sublayers.

`- setNeedsLayout` (page 79)
: Called when the preferred size of the receiver may have changed.

`constraints` (page 51) *property*
: Specifies the constraints used to layout the receiver's sublayers when using an `CAConstraintManager` instance as the layout manager.

`- addConstraint:` (page 64)
: Adds the constraint to the receiver's array of constraint objects.

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

`autoresizingMask` (page 48) *property*
: A bitmask defining how the layer is resized when the bounds of its superlayer changes.

`- resizeWithOldSuperlayerSize:` (page 77)
: Informs the receiver that the bounds size of its superview has changed.

`- resizeSublayersWithOldSize:` (page 76)
: Informs the receiver's sublayers that the receiver's bounds rectangle size has changed.

`- preferredFrameSize` (page 74)
: Returns the preferred frame size of the layer in the coordinate space of the superlayer.

`- layoutIfNeeded` (page 73)
: Recalculate the receiver's layout, if required.

`- layoutSublayers` (page 73)
: Called when the layer requires layout.

Actions

`actions` (page 47) *property*
: A dictionary mapping keys to objects that implement the `CAAction` protocol.

`+ defaultActionForKey:` (page 61)
: Returns an object that implements the default action for the specified identifier.

- `actionForKey:` (page 63)
  Returns an object that implements the action for the specified identifier.
- `actionForLayer:forKey:` (page 79) *delegate method*
  Allows the delegate to customize the action for a layer.

Mapping Between Coordinate and Time Spaces

- `convertPoint:fromLayer:` (page 66)
  Converts the point from the specified layer's coordinate system to the receiver's coordinate system.
- `convertPoint:toLayer:` (page 67)
  Converts the point from the receiver's coordinate system to the specified layer's coordinate system.
- `convertRect:fromLayer:` (page 67)
  Converts the rectangle from the specified layer's coordinate system to the receiver's coordinate system.
- `convertRect:toLayer:` (page 67)
  Converts the rectangle from the receiver's coordinate system to the specified layer's coordinate system.
- `convertTime:fromLayer:` (page 68)
  Converts the time interval from the specified layer's time space to the receiver's time space.
- `convertTime:toLayer:` (page 68)
  Converts the time interval from the receiver's time space to the specified layer's time space

Hit Testing

- `hitTest:` (page 70)
  Returns the farthest descendant of the receiver in the layer hierarchy (including itself) that contains a specified point.
- `containsPoint:` (page 66)
  Returns whether the receiver contains a specified point.

Rendering

- `renderInContext:` (page 75)
  Renders the receiver and its sublayers into the specified context.

Scrolling

`visibleRect` (page 61) *property*
  Returns the visible region of the receiver, in its own coordinate space. (read-only)
- `scrollPoint:` (page 77)
  Scrolls the receiver's closest ancestor `CAScrollLayer` so that the specified point lies at the origin of the layer.
- `scrollRectToVisible:` (page 77)
  Scrolls the receiver's closest ancestor `CAScrollLayer` the minimum distance needed so that the specified rectangle becomes visible.

Modifying the Delegate

`delegate` (page 52) *property*
Specifies the receiver's delegate object.

Key-Value Coding Extensions

`– shouldArchiveValueForKey:` (page 79)
Specifies whether the value of the property for a given key is archived.

`+ defaultValueForKey:` (page 62)
Specifies the default value of the property with the specified key.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 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 63) for a description of the action search pattern.

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

See Also
`– actionForKey:` (page 63)
`– actionForLayer:forKey:` (page 79)
`+ defaultActionForKey:` (page 61)
`@property style` (page 59)

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. Defaults to (0.5, 0.5), the center of the bounds rectangle.

See Layer Geometry and Transforms in *Core Animation Programming Guide* for more information on the relationship between the `bounds` (page 50), `anchorPoint` (page 47) and `position` (page 57) properties.

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

See Also
`@property position` (page 57)

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 81) for possible values. Default value is `kCALayerNotSizable` (page 81).

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

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.

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 iPhone OS. 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.

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 51) and `sublayers` (page 60) and includes the effects of the `cornerRadius` (page 52) property. The default is 0.0.

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

Declared In
`CALayer.h`

bounds

Specifies the bounds rectangle of the receiver. Animatable.

```
@property CGRect bounds
```

Discussion
The default is an empty rectangle.

See Layer Geometry and Transforms in *Core Animation Programming Guide* for more information on the relationship between the `bounds` (page 50), `anchorPoint` (page 47) and `position` (page 57) properties.

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

Declared In
`CALayer.h`

compositingFilter

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

```
@property(retain) CIFilter *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 iPhone OS. Currently the filters available for this property are undefined.

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

See Also
  `@property backgroundFilters` (page 48)

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 NSArray *constraints
```

Discussion
See CAConstraintLayoutManager Class Reference (page 33) 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 52)

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 83). The default value is `kCAGravityResize` (page 85).

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

Declared In
`CALayer.h`

contentsRect

A rectangle, in the unit coordinate space, defining the subrectangle of `contents` (page 51) 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 51)

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.

Declared In
`CALayer.h`

delegate

Specifies the receiver's delegate object.

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

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

Related Sample Code
CALayerEssentials

Declared In
`CALayer.h`

doubleSided

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

```
@property 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.

See Also
`– isDoubleSided` (page 72)

Declared In
`CALayer.h`

edgeAntialiasingMask

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

```
@property unsigned int edgeAntialiasingMask
```

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

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

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

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

Declared In
`CALayer.h`

filters

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

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

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

Special Considerations

While the `CALayer` class exposes this property, Core Image is not available in iPhone OS. 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 50), `anchorPoint` (page 47) and `position` (page 57) properties. When the `frame` is set, the receiver's `position` (page 57) and the size of the receiver's `bounds` (page 50) are changed to match the new frame rectangle.

See Layer Geometry and Transforms in *Core Animation Programming Guide* for more information on the relationship between the `bounds` (page 50), `anchorPoint` (page 47) and `position` (page 57) properties.

Note: The `frame` property is not directly animatable. Instead you should animate the appropriate combination of the `bounds` (page 50), `anchorPoint` (page 47) and `position` (page 57) properties to achieve the desired result.

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

Related Sample Code
CALayerEssentials

Declared In
`CALayer.h`

hidden

Determines whether the receiver is displayed. Animatable.

```
@property BOOL hidden
```

Discussion
The default is `NO`.

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

See Also
`– isHidden` (page 72)

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. 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 85). The default value is `kCAFilterLinear` (page 86).

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 52) property. If `YES` and a `mask` (page 55) 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.

Declared In
`CALayer.h`

minificationFilter

The filter used when reducing the size of the content.

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

Discussion
The possible values for `minifcationFilter` are shown in "`Scaling Filters`" (page 85). The default value is `kCAFilterLinear` (page 86).

Availability
Available in Mac OS X v10.5 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.

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 78) is automatically invoked when the receiver's `bounds` (page 50) 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.

Declared In
`CALayer.h`

opaque

Specifies a hint marking that the pixel data provided by the `contents` (page 51) property is completely opaque.

```
@property BOOL opaque
```

Discussion
Defaults to `NO`.

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

See Also
`– isOpaque` (page 73)

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 47). The default is (0.0,0.0).

See Layer Geometry and Transforms in *Core Animation Programming Guide* for more information on the relationship between the `bounds` (page 50), `anchorPoint` (page 47) and `position` (page 57) properties.

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

See Also
`@property anchorPoint` (page 47)

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.

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.

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:` 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.

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 60) set to `nil`.

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

Declared In
`CALayer.h`

sublayerTransform

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

```
@property CATransform3D sublayerTransform
```

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

Availability
Available in 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.

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.

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.

Declared In
`CAScrollLayer.h`

zPosition

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

```
@property CGFloat zPosition
```

Discussion
Defaults to 0.

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

Declared In
`CALayer.h`

Class Methods

defaultActionForKey:

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

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

Parameters

aKey

The identifier of the action.

Return Value

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

Discussion

See `actionForKey:` (page 63) for a description of the action search pattern.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `actionForKey:` (page 63)
- `actionForLayer:forKey:` (page 79)

`@property actions` (page 47)

`@property style` (page 59)

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
Core Animation QuickTime Layer

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:

- If defined, return the object provided by the receiver's delegate method `actionForLayer:forKey:` (page 79).
- Return the object that corresponds to the identifier in the receiver's `actions` (page 47) dictionary property.
- Search the `style` (page 59) dictionary recursively for an actions dictionary that contains the identifier.
- Call the receiver's `defaultActionForKey:` (page 61) method and return the result.

If any of these steps results in a non-`nil` action object, the remaining steps are ignored and the action is returned. If a step returns an `NSNull` object, the remaining steps are ignored and `nil` is returned.

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 79)
`@property actions` (page 47)
`+ defaultActionForKey:` (page 61)
`@property style` (page 59)

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 82) is automatically used for transition animations. The `nil` pointer is also a valid key.

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

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

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 33) for more information.

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

Declared In
`CAConstraintLayoutManager.h`

addSublayer:

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

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

Parameters

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

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

Related Sample Code
CALayerEssentials

Declared In
`CALayer.h`

affineTransform

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

```
- (CGAffineTransform)affineTransform
```

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

Availability
Available in 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`

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`

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.

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* and 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.

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 69) method, then updates the receiver's `contents` (page 51) property. You should not call this method directly.

Subclasses can override this method to set the `contents` (page 51) property to an appropriate `CGImageRef`.

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

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 69) method when the `contents` (page 51) property is being updated.

Subclasses can override this method to draw the receiver's content.

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 63)

Declared In
`CALayer.h`

initWithLayer:

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

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

Parameters

layer
The layer from which custom fields should be copied.

Return Value
A layer instance with any custom instance variables copied from *layer*.

Discussion
This initializer is used to create shadow copies of layers, for example, for the presentationLayer method.

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

Subclasses should always invoke the superclass implementation

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

Availability
Available in 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 60) 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

Declared In
`CALayer.h`

insertSublayer:below:

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

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

Parameters

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

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

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

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

Declared In
`CALayer.h`

isDoubleSided

A synthesized accessor for the `doubleSided` (page 53) property.

```
- (BOOL)isDoubleSided
```

See Also
`@property doubleSided` (page 53)

isHidden

A synthesized accessor for the `hidden` (page 54) property.

```
- (BOOL)isHidden
```

See Also
`@property hidden` (page 54)

isOpaque

A synthesized accessor for the `opaque` (page 57) property.

```
- (BOOL)isOpaque
```

See Also

`@property opaque` (page 57)

layoutIfNeeded

Recalculate the receiver's layout, if required.

```
- (void)layoutIfNeeded
```

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:` (page 260), if a layout manager is specied 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.

Declared In

`CALayer.h`

modelLayer

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

```
- (id)presentationLayer
```

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`

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 50) rectangle mapped into coordinate space of the receiver's `superlayer` (page 60).

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 60), `mask` (page 55), and `superlayer` (page 60) 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 70) 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.

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 60) array or `mask` (page 55) property of the receiver's `superlayer` (page 60).

```
- (void)removeFromSuperlayer
```

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

Declared In
`CALayer.h`

renderInContext:

Renders the receiver and its sublayers into the specified context.

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

Parameters

ctx

The graphics context that the content is rendered in to.

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

Important: The Mac OS X v10.5 implementation of this method does not support the entire Core Animation composition model. `QCCompositionLayer`, `CAOpenGLLayer`, and `QTMovieLayer` layers are not rendered. Additionally, layers that use 3D transforms are not rendered, nor are layers that specify `backgroundFilters` (page 48), `filters` (page 53), `compositingFilter` (page 50), or a `mask` (page 55) 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 61) property as an affine transform.

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

Parameters

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

Availability
Available in 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 69) which may result in the delegate receiving either a `displayLayer:` (page 80) or `drawLayer:inContext:` (page 80) message.

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

Related Sample Code
CALayerEssentials

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 73) on the receiver and all its superlayers). If the receiver's layout manager implements the `invalidateLayoutOfLayer:` (page 259) 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 *)key
```

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 63) for a description of the action search pattern.

Availability

Available in Mac OS X v10.5 and later.

See Also

`- actionForLayer:forKey:` (page 79)
`@property actions` (page 47)
`+ defaultActionForKey:` (page 61)
`@property style` (page 59)

Declared In

`CALayer.h`

displayLayer:

Allows the delegate to override the `display` (page 69) 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 69).

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 48) 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 63), `addAnimation:forKey:` (page 64), `defaultActionForKey:` (page 61), `removeAnimationForKey:` (page 75), `actionForLayer:forKey:` (page 79), and the `CAAction` protocol method `runActionForKey:object:arguments:` (page 257).

```
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 53) 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 51) 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 verticallycentered 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 55) and `minificationFilter` (page 56).

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

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`.

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

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

Overview

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

Tasks

Creating Timing Functions

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

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

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

Accessing the Control Points

`- getControlPointAtIndex:values:` (page 91)
Returns the control point for the specified index.

Class Methods

functionWithControlPoints::::

Creates and returns a new instance of `CAMediaTimingFunction` timing function modeled as a cubic bezier 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 bezier curve are automatically set to (0.0,0.0) and (1.0,1.0). The control points defining the bezier 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 92).

Return Value

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

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CAMediaTimingFunction.h`

Instance Methods

getControlPointAtIndex:values:

Returns the control point for the specified index.

```
- (void)getControlPointAtIndex:(size_t)index
    values:(float[2])ptr
```

Parameters

index

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

ptr

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

Discussion

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

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CAMediaTimingFunction.h`

initWithControlPoints::::

Returns an initialized timing function modeled as a cubic bezier 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 bezier curve are automatically set to (0.0,0.0) and (1.0,1.0). The control points defining the bezier 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 90).

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

Constants

`kCAMediaTimingFunctionLinear`
Specifies linear pacing. A 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`.

Declared In
`CAMediaTimingFunction.h`

CAOpenGLLayer Class Reference

Inherits from	CALayer : NSObject
Conforms to	NSCoding (CALayer) CAMediaTiming (CALayer) NSObject (NSObject)
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	CAOpenGLLayer.h
Companion guides	Core Animation Programming Guide Core Animation Cookbook
Related sample code	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 98). You can specify that the OpenGL content is static by setting the `asynchronous` (page 96) property to `NO`.

Tasks

Drawing the Content

`asynchronous` (page 96) *property*
Determines when the contents of the layer are updated.

– `isAsynchronous` (page 99)
A synthesized accessor for the `asynchronous` (page 96) property.

– `canDrawInCGLContext:pixelFormat:forLayerTime:displayTime:` (page 97)
Returns whether the receiver should draw OpenGL content for the specified time.

– `drawInCGLContext:pixelFormat:forLayerTime:displayTime:` (page 98)
Draws the OpenGL content for the specified time.

Managing the Pixel Format

- `copyCGLPixelFormatForDisplayMask:` (page 98)
  Returns the OpenGL pixel format suitable for rendering to the set of displays specified by the display mask.
- `releaseCGLPixelFormat:` (page 99)
  Releases the specified OpenGL pixel format object.

Managing the Rendering Context

- `copyCGLContextForPixelFormat:` (page 97)
  Returns the rendering context the receiver requires for the specified pixel format.
- `releaseCGLContext:` (page 99)
  Releases the specified rendering context.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 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 78) message. When `YES`, the receiver's `canDrawInCGLContext:pixelFormat:forLayerTime:displayTime:` (page 97) 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 99)

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:(CFTimeInterval)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 pixelf 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 96) property.

```
- (BOOL)isAsynchronous
```

See Also
`@property asynchronous` (page 96)

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 97) 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 97).

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

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

Overview

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

Tasks

Animated Key Path

`keyPath` (page 103) *property*
Specifies the key path the receiver animates.

Property Value Calculation Behavior

`cumulative` (page 102) *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.

`- isCumulative` (page 104)
A synthesized accessor for the `cumulative` (page 102) property.

`additive` (page 102) *property*
: Determines if the value specified by the animation is added to the current render tree value to produce the new render tree value.

`– isAdditive` (page 103)
: A synthesized accessor for the `additive` (page 102) property.

Creating an Animation

`+ animationWithKeyPath:` (page 103)
: Creates and returns an `CAPropertyAnimation` instance for the specified key path.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 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 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 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.

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`

Instance Methods

isAdditive

A synthesized accessor for the `additive` (page 102) property.

```
- (BOOL)isAdditive
```

See Also

`@property additive` (page 102)

isCumulative

A synthesized accessor for the `cumulative` (page 102) property.

```
- (BOOL)isCumulative
```

See Also

`@property cumulative` (page 102)

CARenderer Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	CARenderer.h
Companion guides	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 106) *property*
The root layer of the layer-tree the receiver should render.

Renderer Geometry

`bounds` (page 106) *property*
The bounds of the receiver.

Create a New Renderer

`+ rendererWithCGLContext:options:` (page 107)
Creates and returns a `CARenderer` instance with the render target specified by the Core OpenGL context.

Render a Frame

- `beginFrameAtTime:timeStamp:` (page 107)
  Begin rendering a frame at the specified time.
- `updateBounds` (page 109)
  Returns the bounds of the update region that contains all pixels that will be rendered by the current frame.
- `addUpdateRect:` (page 107)
  Adds the rectangle to the update region of the current frame.
- `render` (page 108)
  Render the update region of the current frame to the target context.
- `nextFrameTime` (page 108)
  Returns the time at which the next update should happen.
- `endFrame` (page 108)
  Release any data associated with the current frame.

Properties

For more about Objective-C properties, see “Properties” in *The Objective-C 2.0 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`

CAScrollLayer Class Reference

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

Overview

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

Tasks

Scrolling Constraints

`scrollMode` (page 112) *property*
Defines the axes in which the layer may be scrolled.

Scrolling the Layer

`– scrollToPoint:` (page 112)
Changes the origin of the receiver to the specified point.

`– scrollToRect:` (page 112)
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 2.0 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 113). The default is `kCAScrollBoth`.

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

Declared In
`CAScrollLayer.h`

Instance Methods

scrollToPoint:

Changes the origin of the receiver to the specified point.

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

Parameters

`thePoint`
The new origin.

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

Declared In
`CAScrollLayer.h`

scrollToRect:

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

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

Parameters

`theRect`
The rectangle that should be visible.

Availability
Available in 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 112) 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`

CATextLayer Class Reference

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

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 disabled 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: When a `CATextLayer` instance is positioned using the *CAConstraintLayoutManager Class Reference* the bounds of the layer is resized to fit the text content.

Tasks

Getting and Setting the Text

`string` (page 118) *property*
The text to be rendered by the receiver.

Text Visual Properties

`font` (page 116) *property*
  The font used to render the receiver's text.

`fontSize` (page 117) *property*
  The font size used to render the receiver's text.

`foregroundColor` (page 117) *property*
  The color used to render the receiver's text.

Text Alignment and Truncation

`wrapped` (page 118) *property*
  Determines whether the text is wrapped to fit within the receiver's bounds.

– `isWrapped` (page 119)
  A synthesized accessor for the `wrapped` (page 118) property.

`alignmentMode` (page 116) *property*
  Determines how individual lines of text are horizontally aligned within the receiver's bounds.

`truncationMode` (page 118) *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 2.0 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 120). Defaults to `kCAAlignmentNatural` (page 120).

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

Declared In
`CATextLayer.h`

font

The font used to render the receiver's text.

```
@property CFTypeRef font
```

Discussion
May be either a `CTFontRef`, a `CGFontRef`, an instance of `NSFont`, or a string naming the font. Defaults to Helvetica.

The `font` property is only used when the `string` (page 118) property is not an `NSAttributedString`.

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

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

Declared In
`CATextLayer.h`

fontSize

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

```
@property CGFloat fontSize
```

Discussion
Defaults to 36.0.

The `font` property is only used when the `string` (page 118) property is not an `NSAttributedString`.

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

Declared In
`CATextLayer.h`

foregroundColor

The color used to render the receiver's text.

```
@property CGColorRef foregroundColor
```

Discussion
Defaults to opaque white.

The `font` property is only used when the `string` (page 118) property is not an `NSAttributedString`.

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

Related Sample Code
CALayerEssentials

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

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 119). Defaults to `kCATruncationNone` (page 119).

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 BOOL wrapped
```

Discussion
Defaults to `NO`.

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

See Also
`– isWrapped` (page 119)

Declared In
`CATextLayer.h`

Instance Methods

isWrapped

A synthesized accessor for the `wrapped` (page 118) property.

```
- (BOOL)isWrapped
```

See Also

`@property wrapped` (page 118)

Constants

Truncation modes

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

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

Constants

`kCATruncationNone`

If the `wrapped` (page 118) 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 116) 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`.

`kCAAlignmentRight`
Text is visually right aligned.

Available in Mac OS X v10.5 and later.

Declared in `CATextLayer.h`.

`kCAAlignmentCenter`
Text is visually center aligned.

Available in Mac OS X v10.5 and later.

Declared in `CATextLayer.h`.

`kCAAlignmentJustified`
Text is justified.

Available in Mac OS X v10.5 and later.

Declared in `CATextLayer.h`.

Declared In
`CATextLayer.h`

CATiledLayer Class Reference

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

Overview

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

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

Regions of the layer may be invalidated using the `setNeedsDisplayInRect:` (page 78) method however the update will be asynchronous. While the next display update will most likely not contain the updated content, a future update will.

Tasks

Visual Fade

`+ fadeDuration` (page 123)
The time, in seconds, that newly added images take to "fade-in" to the rendered representation of the tiled layer.

Levels of Detail

`levelsOfDetail` (page 122) *property*
: The number of levels of detail maintained by this layer.

`levelsOfDetailBias` (page 122) *property*
: The number of magnified levels of detail for this layer.

Layer Tile Size

`tileSize` (page 123) *property*
: The maximum size of each tile used to create the layer's content.

Properties

For more about Objective-C properties, see “Properties” in *The Objective-C 2.0 Programming Language*.

levelsOfDetail

The number of levels of detail maintained by this layer.

```
@property size_t levelsOfDetail
```

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

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

Related Sample Code
CALayerEssentials

Declared In
`CATiledLayer.h`

levelsOfDetailBias

The number of magnified levels of detail for this layer.

```
@property size_t levelsOfDetailBias
```

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

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

Related Sample Code
CALayerEssentials

Declared In
`CATiledLayer.h`

tileSize

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

```
@property CGSize tileSize
```

Discussion
Defaults to (256.0, 256.0).

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

Declared In
`CATiledLayer.h`

Class Methods

fadeDuration

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

```
+ (CFTimeInterval)fadeDuration
```

Discussion
The default implementation returns 0.25 seconds.

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

Declared In
`CATiledLayer.h`

CATransaction Class Reference

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

Overview

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

Core Animation supports two types of transactions: *implicit* transactions and *explicit* transactions. Implicit transactions are created automatically when the layer tree is modified by a thread without an active transaction and are committed automatically when the thread's run-loop next iterates. Explicit transactions occur when the the application sends the CATransaction class a `begin` (page 126) message before modifying the layer tree, and a `commit` (page 126) message afterwards.

In some circumstances (for example, if there is no run-loop, or the run-loop is blocked) it may be necessary to use explicit transactions to get timely render tree updates.

Tasks

Creating and Committing Transactions

`+ begin` (page 126)
Begin a new transaction for the current thread.

`+ commit` (page 126)
Commit all changes made during the current transaction.

`+ flush` (page 126)
Flushes any extant implicit transaction.

Getting and Setting Transaction Properties

`+ valueForKey:` (page 127)
  Returns the arbitrary keyed-data specified by the given key.

`+ setValue:forKey:` (page 127)
  Sets the arbitrary keyed-data for the specified key.

Class Methods

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.

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.

Declared In
`CATransaction.h`

flush

Flushes any extant implicit transaction.

```
+ (void)flush
```

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

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

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.

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 127) and `setValue:forKey:` (page 127).

```
NSString * const kCATransactionAnimationDuration;
NSString * const kCATransactionDisableActions;
```

Constants

`kCATransactionAnimationDuration`

Default duration, in seconds, for animations added to layers. 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 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`.

Declared In

`CATransaction.h`

CATransition Class Reference

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

Overview

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

Tasks

Transition Start and End Point

`startProgress` (page 131) *property*
Indicates the start point of the receiver as a fraction of the entire transition.

`endProgress` (page 130) *property*
Indicates the end point of the receiver as a fraction of the entire transition.

Transition Properties

`type` (page 131) *property*
Specifies the predefined transition type.

`subtype` (page 131) *property*
Specifies an optional subtype that indicates the direction for the predefined motion-based transitions.

Custom Transition Filter

`filter` (page 130) *property*
: An optional CoreImage filter object that provides the transition.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 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 131), and not greater than 1.0. If `endProgress` is less than `startProgress` (page 131) 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 CoreImage 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 131) and `subtype` (page 131) properties are ignored.

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 132). The default is `nil`.

This property is ignored if a custom transition is specified in the `filter` (page 130) 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 132). This property is ignored if a custom transition is specified in the `filter` (page 130) property. The default is `kCATransitionFade` (page 132).

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 131) 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 132) 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 132) 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 132) 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 131) 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`

CIColor Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	Library/Frameworks/QuartzCore.framework
Declared in	QuartzCore/CIColor.h
Availability	Mac OS X v10.4 and later
Companion guides	Core Image Programming Guide Color Management Overview
Related sample code	CIAnnotation

Overview

The `CIColor` class contains color values and the color space for which the color values are valid. You use `CIColor` objects in conjunction with other Core Image classes, such as `CIFilter`, `CIContext`,and `CIImage`, to take advantage of the built-in Core Image filters when processing images.

A color space defines a one-, two-, three-, or four-dimensional environment whose color components represent intensity values. A color component is also referred to as a color channel. An RGB color space, for example, is a three-dimensional color space whose stimuli are the red, green, and blue intensities that make up a given color. Regardless of the color space, in Core Image, color values range from `0.0` to `1.0`, with `0.0` representing an absence of that component (0 percent) and `1.0` representing 100 percent.

Colors also have an alpha component that represents the opacity of the color, with `0.0` meaning completely transparent and `1.0` meaning completely opaque. If a color does not have an explicit alpha component, Core Image paints the color as if the alpha component equals `1.0`. You always provide unpremultiplied color components to Core Image and Core Image provides unpremultiplied color components to you. Core Image premultiplies each color component with the alpha value in order to optimize calculations. For more information on premultiplied alpha values see *Core Image Programming Guide*.

Tasks

Initializing Color Objects

`– initWithCGColor:` (page 141)
Initializes a color object with a Quartz color.

Creating Color Objects

`+ colorWithCGColor:` (page 137)
Creates a color object from a Quartz color.

`+ colorWithRed:green:blue:` (page 137)
Creates a color object using the specified RGB color component values

`+ colorWithRed:green:blue:alpha:` (page 138)
Creates a color object using the specified RGBA color component values.

`+ colorWithString:` (page 139)
Creates a color object using the RGBA color component values specified by a string.

Getting Color Components

`– alpha` (page 139)
Returns the alpha value of the color.

`– blue` (page 140)
Returns the blue component of the color.

`– colorSpace` (page 140)
Returns the Quartz 2D color space associated with the color.

`– components` (page 140)
Returns the color components of the color.

`– green` (page 141)
Returns the green component of the color.

`– numberOfComponents` (page 141)
Returns the number of color components in the color.

`– red` (page 142)
Returns the red component of the color.

`– stringRepresentation` (page 142)
Returns a formatted string that specifies the components of the color.

Class Methods

colorWithCGColor:

Creates a color object from a Quartz color.

```
+ (CIColor *)colorWithCGColor:(CGColorRef)c
```

Parameters

c

A Quartz color (`CGColorRef` object) created using a Quartz color creation function such as `CGColorCreate`.

Return Value
A Core Image color object that represents a Quartz color.

Discussion
A `CGColorRef` object is the fundamental opaque data type used internally by Quartz to represent colors. For more information on Quartz 2D color and color spaces, see *Quartz 2D Programming Guide*.

You can pass a `CGColorRef` object that represents any color space, including CMYK, but Core Image converts all color spaces to the Core Image working color space before it passes the color space to the filter kernel. The Core Image working color space uses three color components plus alpha.

Availability
Mac OS X v10.4 and later.

See Also
`+ colorWithRed:green:blue:` (page 137)
`+ colorWithRed:green:blue:alpha:` (page 138)
`+ colorWithString:` (page 139)

Declared In
`CIColor.h`

colorWithRed:green:blue:

Creates a color object using the specified RGB color component values

```
+ (CIColor *)colorWithRed:(CGFloat)r green:(CGFloat)g blue:(CGFloat)b
```

Parameters

r

The value of the red component.

g

The value of the green component.

b

The value of the blue component.

Return Value
A Core Image color object that represents an RGB color in the color space specified by the Quartz 2D constant `kCGColorSpaceGenericRGB`.

Availability
Mac OS X v10.4 and later.

See Also
+ `colorWithCGColor:` (page 137)
+ `colorWithRed:green:blue:alpha:` (page 138)
+ `colorWithString:` (page 139)

Declared In
`CIColor.h`

colorWithRed:green:blue:alpha:

Creates a color object using the specified RGBA color component values.

```
+ (CIColor *)colorWithRed:(CGFloat)r green:(CGFloat)g blue:(CGFloat)b
    alpha:(CGFloat)a
```

Parameters

r
The value of the red component.

g
The value of the green component.

b
The value of the blue component.

a
The value of the alpha component.

Return Value
A Core Image color object that represents an RGB color in the color space specified by the Quartz 2D constant `kCGColorSpaceGenericRGB` and an alpha value.

Availability
Mac OS X v10.4 and later.

See Also
+ `colorWithCGColor:` (page 137)
+ `colorWithRed:green:blue:` (page 137)
+ `colorWithString:` (page 139)

Related Sample Code
CIAnnotation

Declared In
`CIColor.h`

colorWithString:

Creates a color object using the RGBA color component values specified by a string.

```
+ (CIColor *)colorWithString:(NSString *)representation
```

Parameters

representation

A string that is in one of the formats returned by the `stringRepresentation` method. For example, the string:

```
@"0.5 0.7 0.3 1.0"
```

indicates an RGB color whose components are 50% red, 70% green, 30% blue, and 100% opaque (alpha value of 1.0). The string representation always has four components—red, green, blue, and alpha. The default value for the alpha component is `1.0`.

Return Value

A Core Image color object that represents an RGB color in the color space specified by the Quartz 2D constant `kCGColorSpaceGenericRGB`.

Availability

Mac OS X v10.4 and later.

See Also

`+ colorWithCGColor:` (page 137)
`+ colorWithRed:green:blue:` (page 137)
`+ colorWithRed:green:blue:alpha:` (page 138)

Declared In

`CIColor.h`

Instance Methods

alpha

Returns the alpha value of the color.

```
- (CGFloat)alpha
```

Return Value

The alpha value. A color created without an explicit alpha value has an alpha of 1.0 by default.

Availability

Mac OS X v10.4 and later.

See Also

`- components` (page 140)

Declared In

`CIColor.h`

blue

Returns the blue component of the color.

```
- (CGFloat)blue
```

Return Value
The unpremultiplied blue component of the color.

Availability
Mac OS X v10.4 and later.

See Also
`– components` (page 140)

Declared In
`CIColor.h`

colorSpace

Returns the Quartz 2D color space associated with the color.

```
- (CGColorSpaceRef)colorSpace
```

Return Value
The Quartz 2D color space (`CGColorSpaceRef` object). You are responsible for disposing of this color space by calling the Quartz 2D function `CGColorSpaceRelease`.

Availability
Mac OS X v10.4 and later.

See Also
`– components` (page 140)

Declared In
`CIColor.h`

components

Returns the color components of the color.

```
- (const CGFloat *)components
```

Return Value
An array of color components, specified as floating-point values in the range of 0.0 through 1.0. This array includes an alpha component if there is one.

Availability
Mac OS X v10.4 and later.

See Also
`– numberOfComponents` (page 141)
`– stringRepresentation` (page 142)

Declared In
`CIColor.h`

green

Returns the green component of the color.

```
- (CGFloat)green
```

Return Value
The unpremultiplied green component of the color.

Availability
Mac OS X v10.4 and later.

See Also
`- components` (page 140)

Declared In
`CIColor.h`

initWithCGColor:

Initializes a color object with a Quartz color.

```
- (id)initWithCGColor:(CGColorRef)c
```

Parameters
c
A Quartz color (`CGColorRef`) created using a Quartz color creation function such as `CGColorCreate`.

Discussion
A `CGColorRef` object is the fundamental opaque data type used internally by Quartz to represent colors. For more information on Quartz 2D color and color spaces, see *Quartz 2D Programming Guide*.

You can pass a `CGColorRef` object that represents any color space, including CMYK, but Core Image converts all color spaces to the Core Image working color space before it passes the color space to the filter kernel. The Core Image working color space uses three color components plus alpha.

Availability
Mac OS X v10.4 and later.

Declared In
`CIColor.h`

numberOfComponents

Returns the number of color components in the color.

```
- (size_t)numberOfComponents
```

Return Value
The number of color components, which includes an alpha component if there is one.

Availability
Mac OS X v10.4 and later.

See Also
`- components` (page 140)

Declared In
`CIColor.h`

red

Returns the red component of the color.

```
- (CGFloat)red
```

Return Value
The unpremultiplied red component of the color.

Availability
Mac OS X v10.4 and later.

See Also
`- components` (page 140)

Declared In
`CIColor.h`

stringRepresentation

Returns a formatted string that specifies the components of the color.

```
- (NSString *)stringRepresentation
```

Return Value
The formatted string.

Discussion
The string representation always has four components—red, green, blue, and alpha. The default value for the alpha component is `1.0`.F or example, this string:

```
@"0.5 0.7 0.3 1.0"
```

indicates an RGB color whose components are 50% red, 70% green, 30% blue, and 100% opaque (alpha value of 1.0).

Availability
Mac OS X v10.4 and later.

See Also
`- components` (page 140)

Declared In
`CIColor.h`

CIContext Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	Library/Frameworks/QuartzCore.framework
Declared in	QuartzCore/CIContext.h
Availability	Mac OS X v10.4 and later
Companion guides	Core Image Programming Guide Image Unit Tutorial
Related sample code	CIAnnotation Reducer UnsharpMask WebKitCIPlugIn WhackedTV

Overview

The `CIContext` class provides an evaluation context for rendering a `CIImage` object through Quartz 2D or OpenGL. You use `CIContext` objects in conjunction with other Core Image classes, such as `CIFilter`, `CIImage`, and `CIColor`, to take advantage of the built-in Core Image filters when processing images.

Tasks

Creating a Context

`+ contextWithCGContext:options:` (page 144)
Creates a Core Image context from a Quartz context, using the specified options.

`+ contextWithCGLContext:pixelFormat:options:` (page 145)
Creates a Core Image context from a CGL context, using the specified options and pixel format object.

Rendering Images

- `createCGImage:fromRect:` (page 146)
  Creates a Quartz 2D image from a region of a `CIImage` object.
- `createCGImage:fromRect:format:colorSpace:` (page 147)
  Creates a Quartz 2D image from a region of a `CIImage` object.
- `createCGLayerWithSize:info:` (page 148)
  Creates a CGLayer object from the provided parameters.
- `drawImage:atPoint:fromRect:` (page 149)
  Renders a region of an image to a point in the context destination.
- `drawImage:inRect:fromRect:` (page 149)
  Renders a region of an image to a rectangle in the context destination.
- `render:toBitmap:rowBytes:bounds:format:colorSpace:` (page 150)
  Renders to the given bitmap.

Managing Resources

- `clearCaches` (page 146)
  Frees any cached data, such as temporary images, associated with the context and runs the garbage collector.
- `reclaimResources` (page 150)
  Runs the garbage collector to reclaim any resources that the context no longer requires.

Class Methods

contextWithCGContext:options:

Creates a Core Image context from a Quartz context, using the specified options.

```
+ (CIContext *)contextWithCGContext:(CGContextRef)ctx options:(NSDictionary *)dict
```

Parameters

`ctx`
A Quartz graphics context (`CGContextRef` object) either obtained from the system or created using a Quartz function such as `CGBitmapContextCreate`. See *Quartz 2D Programming Guide* for information on creating Quartz graphics contexts.

`dict`
A dictionary that contains color space information. You can provide the keys `kCIContextOutputColorSpace` (page 151) or `kCIContextWorkingColorSpace` (page 151) along with a `CGColorSpaceRef`object for each color space.

Discussion

After calling this method, Core Image draws content to the specified Quartz graphics context.

When you create a `CIContext` object using a Quartz graphics context, any transformations that are already set on the Quartz graphics context affect drawing to that context.

Availability
Mac OS X v10.4 and later.

See Also
`+ contextWithCGLContext:pixelFormat:options:` (page 145)

Related Sample Code
CIAnnotation
UnsharpMask

Declared In
`CIContext.h`

contextWithCGLContext:pixelFormat:options:

Creates a Core Image context from a CGL context, using the specified options and pixel format object.

```
+ (CIContext *)contextWithCGLContext:(CGLContextObj)ctx
    pixelFormat:(CGLPixelFormatObj)pf options:(NSDictionary *)dict
```

Parameters

ctx

A CGL context (`CGLContextObj` object) obtain by calling the CGL function `CGLCreateContext`.

pf

A CGL pixel format object (`CGLPixelFormatObj` object) created by calling the CGL function `CGLChoosePixelFormat`. This argument must be the same pixel format object used to create the CGL context. The pixel format object must be valid for the lifetime of the Core Image context. Don't release the pixel format object until after you release the Core Image context.

options

A dictionary that contains color space information. You can provide the keys `kCIContextOutputColorSpace` (page 151) or `kCIContextWorkingColorSpace` (page 151) along with a `CGColorSpaceRef` object for each color space.

Discussion
After calling this method, Core Image draws content into the surface (drawable object) attached to the CGL context. A CGL context is an Mac OS X OpenGL context. For more information, see *OpenGL Programming Guide for Mac OS X*.

When you create a `CIContext` object using a CGL context, all OpenGL states set for the CGL context affect rendering to that context. That means that coordinate and viewport transformations set on the CGL context as well as the vertex color.

For best results, follow these guidelines when you use Core Image to render into an OpenGL context:

- Ensure that the a single unit in the coordinate space of the OpenGL context represents a single pixel in the output device.
- The Core Image coordinate space has the origin in the bottom left corner of the screen. You should configure the OpenGL context in the same way.
- The OpenGL context blending state is respected by Core Image. If the image you want to render contains translucent pixels, it's best to enable blending using a blend function with the parameters `GL_ONE`, `GL_ONE_MINUS_SRC_ALPHA`, as shown in the following code example.

Some typical initialization code for a view with width `W` and height `H` is:

```
glViewport (0, 0, W, H);
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
glOrtho (0, W, 0, H, -1, 1);
glMatrixMode (GL_MODELVIEW);
glLoadIdentity ();
glBlendFunc (GL_ONE, GL_ONE_MINUS_SRC_ALPHA);
glEnable (GL_BLEND);
```

Availability
Mac OS X v10.4 and later.

See Also
`+ contextWithCGContext:options:` (page 144)

Related Sample Code
CIVideoDemoGL

QTCoreImage101

VideoViewer

WebKitCIPlugIn

WhackedTV

Declared In
`CIContext.h`

Instance Methods

clearCaches

Frees any cached data, such as temporary images, associated with the context and runs the garbage collector.

```
- (void)clearCaches
```

Discussion
You can use this method to remove textures from the texture cache that reference deleted images.

Availability
Mac OS X v10.4 and later.

See Also
`- reclaimResources` (page 150)

Declared In
`CIContext.h`

createCGImage:fromRect:

Creates a Quartz 2D image from a region of a `CIImage` object.

```
- (CGImageRef)createCGImage:(CIImage *)im fromRect:(CGRect)r
```

Parameters

im

A `CIImage` object.

r

The region of the image to render.

Return Value

A Quartz 2D (`CGImageRef`) image. You are responsible for releasing the returned image when you no longer need it.

Discussion

Renders a region of an image into a temporary buffer using the context, then creates and returns a Quartz 2D image with the results.

Availability

Mac OS X v10.4 and later.

See Also

`- createCGImage:fromRect:format:colorSpace:` (page 147)

Related Sample Code

CIAnnotation

Declared In

`CIContext.h`

createCGImage:fromRect:format:colorSpace:

Creates a Quartz 2D image from a region of a `CIImage` object.

```
- (CGImageRef)createCGImage:(CIImage *)im fromRect:(CGRect)r
        format:(CIFormat)f colorSpace:(CGColorSpaceRef)cs
```

Parameters

im

A `CIImage` object.

r

The region of the image to render.

f

The format of the image.

cs

The color space of the image.

Return Value

A Quartz 2D (`CGImageRef`) image. You are responsible for releasing the returned image when you no longer need it.

Discussion

Renders a region of an image into a temporary buffer using the context, then creates and returns a Quartz 2D image with the results.

Availability
Mac OS X v10.5 and later.

See Also
`- createCGImage:fromRect:` (page 146)

Declared In
`CIContext.h`

createCGLayerWithSize:info:

Creates a CGLayer object from the provided parameters.

```
- (CGLayerRef)createCGLayerWithSize:(CGSize)size info:(CFDictionaryRef)d
```

Parameters

size
: The size, in default user space units, of the layer relative to the graphics context.

d
: A dictionary, which is passed to `CGLayerCreateWithContext` as the `auxiliaryInfo` parameter. Pass `NULL` as this parameter is reserved for future use.

Return Value
A CGLayer (`CGLayerRef`) object.

Discussion
After calling this method, Core Image draws content into the CGLayer object. Core Image creates a CGLayer object by calling the Quartz 2D function `CGLayerCreateWithContext`, whose prototype is:

```
CGLayerRef CGLayerCreateWithContext (
   CGContextRef context,
   CGSize size,
   CFDictionaryRef auxiliaryInfo
);
```

Core Image passes the `CIContext` object as the `context` parameter, the size as the `size` parameter, and the dictionary as the `auxiliaryInfo` parameter. For more information on CGLayer objects, see *Quartz 2D Programming Guide* and *CGLayer Reference*.

Availability
Mac OS X v10.4 and later.

See Also
`+ imageWithCGLayer:` (page 204)
`+ imageWithCGLayer:options:` (page 204)

Related Sample Code
QTCarbonCoreImage101

Declared In
`CIContext.h`

drawImage:atPoint:fromRect:

Renders a region of an image to a point in the context destination.

```
- (void)drawImage:(CIImage *)im atPoint:(CGPoint)p fromRect:(CGRect)src
```

Parameters

im
A `CIImage` object.

p
The point in the context destination to draw to.

src
The region of the image to draw.

Discussion
You can call this method to force evaluation of the result after you apply a filter using one of the methods of the `CIFilter` class, such as `apply:` (page 161), `apply:arguments:options:` (page 161), and `apply:k, . . ..`

Availability
Mac OS X v10.4 and later.

See Also
`- drawImage:inRect:fromRect:` (page 149)

Related Sample Code
QTCarbonCoreImage101
Reducer

Declared In
`CIContext.h`

drawImage:inRect:fromRect:

Renders a region of an image to a rectangle in the context destination.

```
- (void)drawImage:(CIImage *)im inRect:(CGRect)dest fromRect:(CGRect)src
```

Parameters

im
A `CIImage` object.

dest
The rectangle in the context destination to draw into.

src
The subregion of the image that you want to draw into the context, with the origin and target size defined by the `dest` parameter.

Discussion
You can call this method to force evaluation of the result after you you apply a filter using one of the methods of the `CIFilter` class, such as `apply:` (page 161), `apply:arguments:options:` (page 161), and `apply:k, . . ..`

Availability
Mac OS X v10.4 and later.

See Also
`- drawImage:atPoint:fromRect:` (page 149)

Related Sample Code
QTCarbonCoreImage101

Declared In
`CIContext.h`

reclaimResources

Runs the garbage collector to reclaim any resources that the context no longer requires.

```
- (void)reclaimResources
```

Discussion
The system calls this method automatically after every rendering operation. You can use this method to remove textures from the texture cache that reference deleted images.

Availability
Mac OS X v10.4 and later.

See Also
`- clearCaches` (page 146)

Declared In
`CIContext.h`

render:toBitmap:rowBytes:bounds:format:colorSpace:

Renders to the given bitmap.

```
- (void)render:(CIImage *)im toBitmap:(void *)data rowBytes:(ptrdiff_t)rb
    bounds:(CGRect)r format:(CIFormat)f colorSpace:(CGColorSpaceRef)cs
```

Parameters

im
: A `CIImage` object.

data
: Storage for the bitmap data.

rb
: The bytes per row.

r
: The bounds of the bitmap data.

f
: The format of the bitmap data.

cs

The color space for the data. Pass `NULL` if you want to use the output color space of the context.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CIContext.h`

Constants

Context Options

Keys in the options dictionary for a `CIContext` object.

```
extern NSString *kCIContextOutputColorSpace;
extern NSString *kCIContextWorkingColorSpace;
extern NSString *kCIContextUseSoftwareRenderer;
```

Constants

`kCIContextOutputColorSpace`

A key for the color space to use for images before they are rendered to the context. By default, Core Image uses the GenericRGB color space, which leaves color matching to the system. You can specify a different output color space by providing a Quartz 2D `CGColorSpace` object (`CGColorSpaceRef`). (See *Quartz 2D Programming Guide* for information on creating and using `CGColorSpace` objects.)

`kCIContextWorkingColorSpace`

A key for the color space to use for image operations. By default, Core Image assumes that processing nodes are 128 bits-per-pixel, linear light, premultiplied RGBA floating-point values that use the GenericRGB color space. You can specify a different working color space by providing a Quartz 2D `CGColorSpace` object (`CGColorSpaceRef`). Note that the working color space must be RGB-based. If you have YUV data as input (or other data that is not RGB-based), you can use ColorSync functions to convert to the working color space. (See *Quartz 2D Programming Guide* for information on creating and using `CGColorSpace` objects.)

`kCIContextUseSoftwareRenderer`

A key for enabling software renderer use. If the associated `NSNumber` object is `YES`, then the software renderer is required.

Declared In

`CIContext.h`

CIFilter Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	Library/Frameworks/QuartzCore.framework
Declared in	QuartzCore/CIFilter.h QuartzCore/CIRAWFilter.h
Availability	Mac OS X v10.4 and later
Companion guides	Core Image Programming Guide Image Unit Tutorial Core Image Filter Reference
Related sample code	CarbonCocoaCoreImageTab CIAnnotation CIVideoDemoGL QTCoreImage101 Reducer

Overview

The `CIFilter` class produces a `CIImage` object as output. Typically, a filter takes one or more images as input. Some filters, however, generate an image based on other types of input parameters. The parameters of a `CIFilter` object are set and retrieved through the use of key-value pairs.

You use the `CIFilter` object in conjunction with other Core Image classes, such as `CIImage`, `CIContext`, `CIImageAccumulator`, and `CIColor`, to take advantage of the built-in Core Image filters when processing images, creating filter generators, or writing custom filters.

Tasks

Creating a Filter

`+ filterWithName:` (page 157)
Creates a `CIFilter` object for a specific kind of filter.

`+ filterWithName:keysAndValues:` (page 158)
Creates a `CIFilter` object for a specific kind of filter and initializes the input values.

Creating a Filter from a RAW Image

`+ filterWithImageData:options:` (page 156)
Returns a `CIFilter` object initialized with RAW image data supplied to the method.

`+ filterWithImageURL:options:` (page 157)
Returns a `CIFilter` object initialized with data from a RAW image file.

Accessing Registered Filters

`+ filterNamesInCategories:` (page 155)
Returns an array of all published filter names that match all the specified categories.

`+ filterNamesInCategory:` (page 156)
Returns an array of all published filter names in the specified category.

Registering a Filter

`+ registerFilterName:constructor:classAttributes:` (page 160)
Publishes a custom filter that is not packaged as an image unit.

Getting Filter Parameters and Attributes

`- attributes` (page 162)
Returns a dictionary of key-value pairs that describe the filter.

`- inputKeys` (page 163)
Returns an array that contains the names of the input parameters to the filter.

`- outputKeys` (page 164)
Returns an array that contains the names of the output parameters for the filter.

Setting Default Values

`- setDefaults` (page 164)
Sets all input values for a filter to default values.

Applying a Filter

`- apply:arguments:options:` (page 161)
Produces a *`CIImage`* object by applying arguments to a kernel function and using options to control how the kernel function is evaluated.

`- apply:` (page 161)
Produces a `CIImage` object by applying a kernel function.

Getting Localized Information for Registered Filters

`+ localizedNameForFilterName:` (page 159)
Returns the localized name for the specified filter name.

`+ localizedNameForCategory:` (page 159)
Returns the localized name for the specified filter category.

`+ localizedDescriptionForFilterName:` (page 158)
Returns the localized description of a filter for display in the user interface.

`+ localizedReferenceDocumentationForFilterName:` (page 160)
Returns the location of the localized reference documentation that describes the filter.

Class Methods

filterNamesInCategories:

Returns an array of all published filter names that match all the specified categories.

```
+ (NSArray *)filterNamesInCategories:(NSArray *)categories
```

Parameters

categories
One or more filter categories. Pass `nil` to get all filters in all categories.

Return Value
An array that contains all published filter names that match all the categories specified by the `categories` argument.

Discussion
When you pass more than one filter category, this method returns the intersection of the filters in the categories. For example, if you pass the categories `kCICategoryBuiltIn` (page 172) and `kCICategoryFilterGenerator` (page 172), you obtain all the filters that are members of both the built-in and generator categories. But if you pass in `kCICategoryGenerator` and `kCICategoryStylize` (page 171), you will not get any filters returned to you because there are no filters that are members of both the generator and stylize categories. If you want to obtain all stylize and generator filters, you must call the `filterNamesInCategories:` method for each category separately and then merge the results.

Availability
Mac OS X v10.4 and later.

See Also
`+ filterNamesInCategory:` (page 156)

Related Sample Code
CIAnnotation
CITransitionSelectorSample2

Declared In
`CIFilter.h`

filterNamesInCategory:

Returns an array of all published filter names in the specified category.

```
+ (NSArray *)filterNamesInCategory:(NSString *)category
```

Parameters

`category`

A string object that specifies a filter category.

Return Value
An array that contains all published names of the filter in a category.

Availability
Mac OS X v10.4 and later.

See Also
`+ filterNamesInCategories:` (page 155)

Declared In
`CIFilter.h`

filterWithImageData:options:

Returns a `CIFilter` object initialized with RAW image data supplied to the method.

```
+ (CIFilter *)filterWithImageData:(NSData *)data options:(NSDictionary *)options;
```

Parameters

`data`

The RAW image data to initialize the object with.

`options`

A options dictionary. You can pass any of the keys defined in "`RAW Image Options`" (page 177) along with the appropriate value. You should provide a source type identifier hint key (`kCGImageSourceTypeIdentifierHint`) and the appropriate source type value to help the decoder determine the file type. Otherwise it's possible to obtain incorrect results. See the Discussion for an example

Return Value
A `CIFilter` object.

Discussion
After calling this method, the `CIFilter` object returns a `CIImage` object that is properly processed similar to images retrieved using the `outputImage` key.

Here is an example of adding a source type identifier key-value pair to the options dictionary:

```
[opts setObject:(id)CGImageSourceGetTypeWithExtension ((CFStringRef)[[url path]
 pathExtension])
         forKey:(id)kCGImageSourceTypeIdentifierHint];
```

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

See Also
`+ filterWithImageURL:options:` (page 157)

Declared In
`CIRAWFilter.h`

filterWithImageURL:options:

Returns a `CIFilter` object initialized with data from a RAW image file.

```
+ (CIFilter *)filterWithImageURL:(NSURL *)url options:(NSDictionary *)options;
```

Parameters

`url`
The location of a RAW image file.

`options`
An options dictionary. You can pass any of the keys defined in "`RAW Image Options`" (page 177).

Return Value
A `CIFilter` object.

Discussion
After calling this method, the `CIFilter` object returns a `CIImage` object that is properly processed similar to images retrieved using the `outputImage` key.

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

See Also
`+ filterWithImageData:options:` (page 156)

Declared In
`CIRAWFilter.h`

filterWithName:

Creates a `CIFilter` object for a specific kind of filter.

```
+ (CIFilter *)filterWithName:(NSString *)name
```

Parameters

`name`
The name of the filter.

Return Value
A `CIFilter` object whose input values are undefined.

Discussion
You should call `setDefaults` (page 164) after you call this method or set values individually by calling `setValue:forKey`.

Availability
Mac OS X v10.4 and later.

See Also
`+ filterWithName:keysAndValues:` (page 158)

Related Sample Code
CarbonCocoaCoreImageTab

CIAnnotation

CIVideoDemoGL

QTCoreImage101

Reducer

Declared In
`CIFilter.h`

filterWithName:keysAndValues:

Creates a `CIFilter` object for a specific kind of filter and initializes the input values.

```
+ (CIFilter *)filterWithName:(NSString *)namekeysAndValues:key0, ...
```

Parameters

name

The name of the filter.

key0

A list of key-value pairs to set as input values to the filter. Each key is a constant that specifies the name of the input value to set, and must be followed by a value. You signal the end of the list by passing a `nil` value.

Return Value
A `CIFilter` object whose input values are initialized.

Availability
Mac OS X v10.4 and later.

See Also
`+ filterWithName:` (page 157)

Related Sample Code
CIAnnotation

CITransitionSelectorSample2

Declared In
`CIFilter.h`

localizedDescriptionForFilterName:

Returns the localized description of a filter for display in the user interface.

```
+ (NSString *)localizedDescriptionForFilterName:(NSString *)filterName
```

Parameters

filterName

The filter name.

Return Value

The localized description of the filter.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CIFilter.h`

localizedNameForCategory:

Returns the localized name for the specified filter category.

```
+ (NSString *)localizedNameForCategory:(NSString *)category
```

Parameters

category

A filter category.

Return Value

The localized name for the filter category.

Availability

Mac OS X v10.4 and later.

Declared In

`CIFilter.h`

localizedNameForFilterName:

Returns the localized name for the specified filter name.

```
+ (NSString *)localizedNameForFilterName:(NSString *)filterName
```

Parameters

filterName

A filter name.

Return Value

The localized name for the filter.

Availability

Mac OS X v10.4 and later.

Related Sample Code

QTRecorder

Declared In

`CIFilter.h`

localizedReferenceDocumentationForFilterName:

Returns the location of the localized reference documentation that describes the filter.

```
+ (NSURL *)localizedReferenceDocumentationForFilterName:(NSString *)filterName
```

Parameters

filterName

The filter name.

Return Value

A URL that specifies the location of the localized documentation, or `nil` if the filter does not provide localized reference documentation.

Discussion

The URL can be a local file or a remote document on a web server. Because filters created prior to Mac OS X v10.5 could return `nil`, you should be make sure that your code handles this case gracefully.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CIFilter.h`

registerFilterName:constructor:classAttributes:

Publishes a custom filter that is not packaged as an image unit.

```
+ (void)registerFilterName:(NSString *)name constructor:(id)anObject
    classAttributes:(NSDictionary *)attributes
```

Parameters

name

A string object that specifies the name of the filter you want to publish.

anObject

A constructor object that implements the `filterWithName` method.

attributes

A dictionary that contains the class display name and filter categories attributes along with the appropriate value for each attributes. That is, the `kCIAttributeFilterDisplayName` (page 165) attribute and a string that specifies the display name, and the `kCIAttributeFilterCategories` (page 165) and an array that specifies the categories to which the filter belongs (such as `kCICategoryStillImage` (page 171) and `kCICategoryDistortionEffect` (page 169)). All other attributes for the filter should be returned by the custom `attributes` method implement by the filter.

Discussion

In most cases you don't need to use this method because the preferred way to register a custom filter that you write is to package it as an image unit. You do not need to use this method for a filter packaged as an image unit because you register your filter using the `CIPlugInRegistration` protocol. (See *Core Image Programming Guide* for additional details.)

Availability

Mac OS X v10.4 and later.

Declared In
`CIFilter.h`

Instance Methods

apply:

Produces a `CIImage` object by applying a kernel function.

```
- (CIImage *)apply:(CIKernel *)k, ...
```

Parameters

k

A `CIKernel` object that contains a kernel function.

A list of arguments to supply to the kernel function. The supplied arguments must be type-compatible with the function signature of the kernel function. The list of arguments must be terminated by the `nil` object.

Discussion
For example, if the kernel function has this signature:

```
kernel vec4 brightenEffect (sampler src, float k)
```

You would supply two arguments after the `k` argument to the `apply:k, ..` method. In this case, the first argument must be a sampler and the second a floating-point value. For more information on kernels, see *Core Image Kernel Language Reference*.

Availability
Mac OS X v10.4 and later.

See Also
`- apply:arguments:options:` (page 161)

Declared In
`CIFilter.h`

apply:arguments:options:

Produces a *`CIImage`* object by applying arguments to a kernel function and using options to control how the kernel function is evaluated.

```
- (CIImage *)apply:(CIKernel *)k arguments:(NSArray *)args options:(NSDictionary
    *)dict
```

Parameters

k

A `CIKernel` object that contains a kernel function.

args

The arguments that are type compatible with the function signature of the kernel function.

`dict`
A dictionary that contains options (key-value pairs) to control how the kernel function is evaluated.

Return Value
The `CIImage` object produced by a filter.

Discussion
You can pass any of the following keys in the dictionary:

- `kCIApplyOptionExtent` specifies the size of the produced image. The associated value is a four-element array (`NSArray`) that specifies the x-value of the rectangle origin, the y-value of the rectangle origin, and the width, and height.
- `kCIApplyOptionDefinition` specifies the domain of definition (DOD) of the produces image. The associated value is either a Core Image filter shape or a four-element array (`NSArray`) that specifies a rectangle.
- `kCIApplyOptionUserInfo` specifies to retain the associated object and pass it to any callbacks invoked for that filter.

Availability
Mac OS X v10.4 and later.

See Also
`- apply:` (page 161)

Declared In
`CIFilter.h`

attributes

Returns a dictionary of key-value pairs that describe the filter.

```
- (NSDictionary *)attributes
```

Return Value
A dictionary that contains a key for each input and output parameter for the filter. Each key is a dictionary that contains all the attributes of an input or output parameter.

Discussion
For example, the attributes dictionary for the `CIColorControls` filter contains the following information:

```
CIColorControls:
{
    CIAttributeFilterCategories = (
        CICategoryColorAdjustment,
        CICategoryVideo,
        CICategoryStillImage,
        CICategoryInterlaced,
        CICategoryNonSquarePixels,
        CICategoryBuiltIn
    );
    CIAttributeFilterDisplayName = "Color Controls";
    CIAttributeFilterName = CIColorControls;
    inputBrightness = {
        CIAttributeClass = NSNumber;
        CIAttributeDefault = 0;
        CIAttributeIdentity = 0;
        CIAttributeMin = -1;
        CIAttributeSliderMax = 1;
        CIAttributeSliderMin = -1;
        CIAttributeType = CIAttributeTypeScalar;
    };
    inputContrast = {
        CIAttributeClass = NSNumber;
        CIAttributeDefault = 1;
        CIAttributeIdentity = 1;
        CIAttributeMin = 0.25;
        CIAttributeSliderMax = 4;
        CIAttributeSliderMin = 0.25;
        CIAttributeType = CIAttributeTypeScalar;
    };
    inputImage = {CIAttributeClass = CIImage; };
    inputSaturation = {
        CIAttributeClass = NSNumber;
        CIAttributeDefault = 1;
        CIAttributeIdentity = 1;
        CIAttributeMin = 0;
        CIAttributeSliderMax = 3;
        CIAttributeSliderMin = 0;
        CIAttributeType = CIAttributeTypeScalar;
    };
    outputImage = {CIAttributeClass = CIImage; };
}
```

Availability
Mac OS X v10.4 and later.

Related Sample Code
CITransitionSelectorSample2

Declared In
`CIFilter.h`

inputKeys

Returns an array that contains the names of the input parameters to the filter.

```
- (NSArray *)inputKeys
```

Return Value
An array that contains the names of all input parameters to the filter.

Availability
Mac OS X v10.4 and later.

Related Sample Code
CITransitionSelectorSample2

Declared In
`CIFilter.h`

outputKeys

Returns an array that contains the names of the output parameters for the filter.

```
- (NSArray *)outputKeys
```

Return Value
An array that contains the names of all output parameters from the filter.

Availability
Mac OS X v10.4 and later.

Declared In
`CIFilter.h`

setDefaults

Sets all input values for a filter to default values.

```
- (void)setDefaults
```

Discussion
Input values whose default values are not defined are left unchanged.

Availability
Mac OS X v10.4 and later.

Related Sample Code
CarbonCocoaCoreImageTab
Core Animation QuickTime Layer
QTCarbonCoreImage101
QTRecorder
UnsharpMask

Declared In
`CIFilter.h`

Constants

Filter Attribute Keys

Attributes for a filter and its parameters.

```
extern NSString *kCIAttributeFilterName;
extern NSString *kCIAttributeFilterDisplayName;
extern NSString *kCIAttributeDescription;
extern NSString *kCIAttributeReferenceDocumentation;
extern NSString *kCIAttributeFilterCategories;
extern NSString *kCIAttributeClass;
extern NSString *kCIAttributeType;
extern NSString *kCIAttributeMin;
extern NSString *kCIAttributeMax;
extern NSString *kCIAttributeSliderMin;
extern NSString *kCIAttributeSliderMax;
extern NSString *kCIAttributeDefault;
extern NSString *kCIAttributeIdentity;
extern NSString *kCIAttributeName;
extern NSString *kCIAttributeDisplayName;
```

Constants

`kCIAttributeFilterName`

The filter name, specified as an `NSString` object.

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCIAttributeFilterDisplayName`

The localized version of the filter name that is displayed in the user interface.

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCIAttributeDescription`

The localized description of the filter. This description should inform the end user what the filter does and be short enough to display in the user interface for the filter. It is not intended to be technically detailed.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIAttributeReferenceDocumentation`

The localized reference documentation for the filter. The reference should provide developers with technical details.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIAttributeFilterCategories`

An array of filter category keys that specifies all the categories in which the filter is a member.

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCIAttributeClass`

The class of the input parameter for a filter. If you are writing an image unit (see *Image Unit Tutorial*), Core Image supports only these classes for nonexecutable image units: `CIColor`, `CIVector`, `CIImage`, and `NSNumber` only. Executable image units may have input parameters of any class, but Core Image does not generate an automatic user interface for custom classes (see `CIFilter(IKFilterUIAddition)`).

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCIAttributeType`
: The attribute type.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

`kCIAttributeMin`
: The minimum value for a filter parameter, specified as a floating-point value.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

`kCIAttributeMax`
: The maximum value for a filter parameter, specified as a floating-point value.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

`kCIAttributeSliderMin`
: The minimum value, specified as a floating-point value, to use for a slider that controls input values for a filter parameter.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

`kCIAttributeSliderMax`
: The maximum value, specified as a floating-point value, to use for a slider that controls input values for a filter parameter.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

`kCIAttributeDefault`
: The default value, specified as a floating-point value, for a filter parameter.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

`kCIAttributeIdentity`
: If supplied as a value for a parameter, the parameter has no effect on the input image.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

`kCIAttributeName`
: The name of the attribute.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

`kCIAttributeDisplayName`
: The localized display name of the attribute.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

Discussion

Attribute keys are used for the attribute dictionary of a filter. Most entries in the attribute dictionary are optional. The attribute `CIAttributeFilterName` is mandatory. For a parameter, the attribute `kCIAttributeClass` is mandatory.

A parameter of type `NSNumber` does not necessarily need the attributes `kCIAttributeMin` and `kCIAttributeMax`. These attributes are not present when the parameter has no upper or lower bounds. For example, the Gaussian blur filter has a radius parameter with a minimum of `0` but no maximum value to indicate that all nonnegative values are valid.

Declared In
`CIFilter.h`

Data Type Attributes

Numeric data types.

```
extern NSString *kCIAttributeTypeTime;
extern NSString *kCIAttributeTypeScalar;
extern NSString *kCIAttributeTypeDistance;
extern NSString *kCIAttributeTypeAngle;
extern NSString *kCIAttributeTypeBoolean;
extern NSString *kCIAttributeTypeInteger;
extern NSString *kCIAttributeTypeCount;
```

Constants

`kCIAttributeTypeTime`

A parametric time for transitions, specified as a floating-point value in the range of `0.0` to `1.0`.

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCIAttributeTypeScalar`

A scalar value.

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCIAttributeTypeDistance`

A distance.

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCIAttributeTypeAngle`

An angle.

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCIAttributeTypeBoolean`

A Boolean value.

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCIAttributeTypeInteger`

An integer value.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIAttributeTypeCount`
A positive integer value.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

Declared In
`CIFilter.h`

Vector Quantity Attributes

Vector data types.

```
extern NSString *kCIAttributeTypePosition;
extern NSString *kCIAttributeTypeOffset;
extern NSString *kCIAttributeTypePosition3;
extern NSString *kCIAttributeTypeRectangle
```

Constants
`kCIAttributeTypePosition`
A two-dimensional location in the working coordinate space. (A 2-element vector type.)

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCIAttributeTypeOffset`
An offset. (A 2-element vector type.)

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCIAttributeTypePosition3`
A three-dimensional location in the working coordinate space. (A 3-element vector type.)

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCIAttributeTypeRectangle`
A Core Image vector that specifies the *x* and *y* values of the rectangle origin, and the width (*w*) and height (*h*) of the rectangle. The vector takes the form [*x*, *y*, *w*, *h*]. (A 4-element vector type.)

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

Declared In
`CIFilter.h`

Color Attribute Keys

Color types.

```
extern NSString *kCIAttributeTypeOpaqueColor;
extern NSString *kCIAttributeTypeGradient;
```

Constants

`kCIAttributeTypeOpaqueColor`

A Core Image color (`CIColor` object) that specifies red, green, and blue component values. Use this key for colors with no alpha component. If the key is not present, Core Image assumes color with alpha.

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCIAttributeTypeGradient`

An n-by-1 gradient image used to describe a color ramp.

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

Declared In

`CIFilter.h`

Filter Category Keys

Categories of filters.

```
extern NSString *kCICategoryDistortionEffect;
extern NSString *kCICategoryGeometryAdjustment;
extern NSString *kCICategoryCompositeOperation;
extern NSString *kCICategoryHalftoneEffect;
extern NSString *kCICategoryColorAdjustment;
extern NSString *kCICategoryColorEffect;
extern NSString *kCICategoryTransition;
extern NSString *kCICategoryTileEffect;
extern NSString *kCICategoryGenerator;
extern NSString *kCICategoryReduction;
extern NSString *kCICategoryGradient;
extern NSString *kCICategoryStylize;
extern NSString *kCICategorySharpen;
extern NSString *kCICategoryBlur;
extern NSString *kCICategoryVideo;
extern NSString *kCICategoryStillImage;
extern NSString *kCICategoryInterlaced;
extern NSString *kCICategoryNonSquarePixels;
extern NSString *kCICategoryHighDynamicRange ;
extern NSString *kCICategoryBuiltIn;
extern NSString *kCICategoryFilterGenerator;
```

Constants

`kCICategoryDistortionEffect`

A filter that reshapes an image by altering its geometry to create a 3D effect. Using distortion filters, you can displace portions of an image, apply lens effects, make a bulge in an image, and perform other operation to achieve an artistic effect.

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCICategoryGeometryAdjustment`
: A filter that changes the geometry of an image. Some of these filters are used to warp an image to achieve an artistic effects, but these filters can also be used to correct problems in the source image. For example, you can apply an affine transform to straighten an image that is rotated with respect to the horizon.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

`kCICategoryCompositeOperation`
: A filter operates on two image sources, using the color values of one image to operate on the other. Composite filters perform computations such as computing maximum values, minimum values, and multiplying values between input images. You can use compositing filters to add effects to an image, crop an image, and achieve a variety of other effects.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

`kCICategoryHalftoneEffect`
: A filter that simulates a variety of halftone screens, to mimic the halftone process used in print media. The output of these filters has the familiar "newspaper" look of the various dot patterns. Filters are typically named after the pattern created by the virtual halftone screen, such as circular screen or hatched screen.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

`kCICategoryColorAdjustment`
: A filter that changes color values. Color adjustment filters are used to eliminate color casts, adjust hue, and correct brightness and contrast. Color adjustment filters do not perform color management; ColorSync performs color management. You can use Quartz 2D to specify the color space associated with an image. For more information, see *Color Management Overview* and *Quartz 2D Programming Guide*.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

`kCICategoryColorEffect`
: A filter that modifies the color of an image to achieve an artistic effect. Examples of color effect filters include filters that change a color image to a sepia image or a monochrome image or that produces such effects as posterizing.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

`kCICategoryTransition`
: A filter that provides a bridge between two or more images by applying a motion effect that defines how the pixels of a source image yield to that of the destination image.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

`kCICategoryTileEffect`
: A filter that typically applies an effect to an image and then create smaller versions of the image (tiles), which are then laid out to create a pattern that's infinite in extent.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

`kCICategoryGenerator`
A filter that generates a pattern, such as a solid color, a checkerboard, or a star shine. The generated output is typically used as input to another filter.

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCICategoryReduction`
A filter that reduces image data. These filters are used to solve image analysis problems.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCICategoryGradient`
A filter that generates a fill whose color varies smoothly. Exactly how color varies depends on the type of gradient—linear, radial, or Gaussian.

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCICategoryStylize`
A filter that makes a photographic image look as if it was painted or sketched. These filters are typically used alone or in combination with other filters to achieve artistic effects.

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCICategorySharpen`
A filter that sharpens images, increasing the contrast between the edges in an image. Examples of sharpen filters are unsharp mask and sharpen luminance.

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCICategoryBlur`
A filter that softens images, decreasing the contrast between the edges in an image. Examples of blur filters are Gaussian blur and zoom blur.

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCICategoryVideo`
A filter that works on video images.

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCICategoryStillImage`
A filter that works on still images.

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCICategoryInterlaced`
A filter that works on interlaced images.

Available in Mac OS X v10.4 and later.

Declared in `CIFilter.h`.

`kCICategoryNonSquarePixels`
: A filter that works on non-square pixels.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

`kCICategoryHighDynamicRange`
: A filter that works on high dynamic range pixels.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

`kCICategoryBuiltIn`
: A filter provided by Core Image. This distinguishes built-in filters from plug-in filters.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

`kCICategoryFilterGenerator`
: A filter created by chaining several filters together and then packaged as a `CIFilterGenerator` object.

  Available in Mac OS X v10.5 and later.

  Declared in `CIFilter.h`.

Declared In
`CIFilter.h`

Options for Applying a Filter

Options that control the application of a Core Image filter.

```
extern NSString *kCIApplyOptionExtent;
extern NSString *kCIApplyOptionDefinition;
extern NSString *kCIApplyOptionUserInfo;
```

Constants

`kCIApplyOptionExtent`
: The size of the produced image. The associated value is a four-element array (`NSArray`) that specifies the x-value of the rectangle origin, the y-value of the rectangle origin, and the width and height.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

`kCIApplyOptionDefinition`
: The domain of definition (DOD) of the produced image. The associated value is either a Core Image filter shape or a four-element array (`NSArray`) that specifies a rectangle.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

`kCIApplyOptionUserInfo`
: Information needed by a callback. The associated value is an object that Core Image will pass to any callbacks invoked for that filter.

  Available in Mac OS X v10.4 and later.

  Declared in `CIFilter.h`.

Declared In
`CIFilter.h`

User Interface Control Options

Sets of controls for various user scenarios.

```
extern NSString *kCIUIParameterSet;
extern NSString *kCIUISetBasic;
extern NSString *kCIUISetIntermediate;
extern NSString *kCIUISetAdvanced;
extern NSString *kCIUISetDevelopment;
```

Constants

`kCIUIParameterSet`

The set of input parameters to use. The associated value can be `kCIUISetBasic` (page 173), `kCIUISetIntermediate` (page 173), `kCIUISetAdvanced` (page 173), or `kCIUISetDevelopment` (page 173).

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIUISetBasic`

Controls that are appropriate for a basic user scenario, that is, the minimum of settings to control the filter.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIUISetIntermediate`

Controls that are appropriate for an intermediate user scenario.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIUISetAdvanced`

Controls that are appropriate for an advanced user scenario.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIUISetDevelopment`

Controls that should be visible only for development purposes.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

Discussion

You can use these constants to specify the controls that you want associated with each user scenario. For example, for a filter that has many input parameters you can choose a small set of input parameters that the typical consumer can control and set the other input parameters to default values. For the same filter, however, you can choose to allow professional customers to control all the input parameters.

Declared In

`CIFIlter.h`

Filter Parameter Keys

Keys for input parameters to filters.

```
extern NSString *kCIOutputImageKey;
extern NSString *kCIInputBackgroundImageKey;
extern NSString *kCIInputImageKey;
extern NSString *kCIInputTimeKey;
extern NSString *kCIInputTransformKey;
extern NSString *kCIInputScaleKey;
extern NSString *kCIInputAspectRatioKey;
extern NSString *kCIInputCenterKey;
extern NSString *kCIInputRadiusKey;
extern NSString *kCIInputAngleKey;
extern NSString *kCIInputRefractionKey;
extern NSString *kCIInputWidthKey;
extern NSString *kCIInputSharpnessKey;
extern NSString *kCIInputIntensityKey;
extern NSString *kCIInputEVKey;
extern NSString *kCIInputSaturationKey;
extern NSString *kCIInputColorKey;
extern NSString *kCIInputBrightnessKey;
extern NSString *kCIInputContrastKey;
extern NSString *kCIInputGradientImageKey;
extern NSString *kCIInputMaskImageKey;
extern NSString *kCIInputShadingImageKey;
extern NSString *kCIInputTargetImageKey;
extern NSString *kCIInputExtentKey;
```

Constants

`kCIOutputImageKey`

A key for the `CIImage` object produced by a filter.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputBackgroundImageKey`

A key for the `CIImage` object to use as a background image.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputImageKey`

A key for the `CIImage` object to use as an input image. For filters that also use a background image, this key refers to the foreground image.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputTimeKey`

A key for z scalar value (`NSNumber`) that specifies a time.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputTransformKey`

A key for an `NSAffineTransform` object that specifies a transformation to apply.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputScaleKey`
A key for a scalar value (`NSNumber`) that specifies the amount of the effect.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputAspectRatioKey`
A key for a scalar value (`NSNumber`) that specifies a ratio.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputCenterKey`
A key for a `CIVector` object that specifies the center of the area, as *x* and *y*- coordinates, to be filtered.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputRadiusKey`
A key for a scalar value (`NSNumber`) that specifies that specifies the distance from the center of an effect.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputAngleKey`
A key for a scalar value (`NSNumber`) that specifies an angle.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputRefractionKey`
A key for a scalar value (`NSNumber`) that specifies the index of refraction of the material (such as glass) used in the effect.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputWidthKey`
A key for a scalar value (`NSNumber`) that specifies the width of the effect.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputSharpnessKey`
A key for a scalar value (`NSNumber`) that specifies the amount of sharpening to apply.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputIntensityKey`
A key for a scalar value (`NSNumber`) that specifies an intensity value.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputEVKey`
A key for a scalar value (`NSNumber`) that specifies how many F-stops brighter or darker the image should be.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputSaturationKey`
A key for a scalar value (`NSNumber`) that specifies the amount to adjust the saturation.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputColorKey`
A key for a `CIColor` object that specifies a color value.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputBrightnessKey`
A key for a scalar value (`NSNumber`) that specifies a brightness level.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputContrastKey`
A key for a scalar value (`NSNumber`) that specifies a contrast level.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputGradientImageKey`
A key for a `CIImage` object that specifies an environment map with alpha. Typically, this image contains highlight and shadow.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputMaskImageKey`
A key for a `CIImage` object to use as a mask.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputShadingImageKey`
A key for a `CIImage` object that specifies an environment map with alpha values. Typically this image contains highlight and shadow.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputTargetImageKey`
A key for a `CIImage` object that is the target image for a transition.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

`kCIInputExtentKey`
A key for a `CIVector` object that specifies a rectangle that defines the extent of the effect.

Available in Mac OS X v10.5 and later.

Declared in `CIFilter.h`.

Discussion
These keys represent some of the most commonly used input parameters. A filter can use other kinds of input parameters.

Declared In
`CIFIlter.h`

RAW Image Options

Options for creating a `CIFilter` object from RAW image data.

```
extern NSString * const kCIInputDecoderVersionKey;
extern NSString * const kCISupportedDecoderVersionsKey;
extern NSString * const kCIInputBoostKey;
extern NSString * const kCIInputNeutralChromaticityXKey;
extern NSString * const kCIInputNeutralChromaticityYKey;
extern NSString * const kCIInputNeutralTemperatureKey;
extern NSString * const kCIInputNeutralTintKey;
extern NSString * const kCIInputNeutralLocation;
extern NSString * const kCIInputScaleFactorKey;
extern NSString * const kCIInputAllowDraftModeKey;
extern NSString * const kCIInputIgnoreImageOrientationKey;
extern NSString * const kCIInputImageOrientationKey;
extern NSString * const kCIInputEnableSharpeningKey;
extern NSString * const kCIInputEnableChromaticNoiseTrackingKey;
extern NSString * const kCIInputBoostShadowAmountKey;
extern NSString * const kCIInputBiasKey;
```

Constants

`kCIInputDecoderVersionKey`

A key for the version number of the method to be used for decoding. A newly initialized object defaults to the newest available decoder version for the given image type. You can request an alternative, older version to maintain compatibility with older releases. Must be one of `kCISupportedDecoderVersions`, otherwise a `nil` output image is generated. The associated value must be an `NSNumber` object that specifies an integer value in range of `0` to the current decoder version. When you request a specific version of the decoder, Core Image produces an image that is *visually* the same across different versions of the operating system. Core Image, however, does not guarantee that the same bits are produced across different versions of the operating system. That's because the rounding behavior of floating-point arithmetic can vary due to differences in compilers or hardware. Note that this option has no effect if the image used for initialization is not RAW.

Available in Mac OS X v10.5 and later.

Declared in `CIRAWFilter.h`.

`kCISupportedDecoderVersionsKey`

A key for the supported decoder versions. The associated value is an `NSArray` object that contains all supported decoder versions for the given image type, sorted in increasingly newer order. Each entry is an `NSDictionary` object that contains key-value pairs. All entries represent a valid version identifier that can be passed as the `kCIDecoderVersion` value for the key `kCIDecoderMethodKey`. Version values are read-only; attempting to set this value raises an exception. Currently, the only defined key is `@"version"` which has as its value an `NSString` that uniquely describing a given decoder version. This string might not be suitable for user interface display..

Available in Mac OS X v10.5 and later.

Declared in `CIRAWFilter.h`.

`kCIInputBoostKey`

A key for the the amount of boost to apply to an image. The associated value is a floating-point value packaged as an `NSNumber` object. The value must be in the range of `0...1`. A value of `0` indicates no boost, that is, a linear response. The default value is `1`, which indicates full boost.

Available in Mac OS X v10.5 and later.

Declared in `CIRAWFilter.h`.

`kCIInputNeutralChromaticityXKey`
: The x value of the chromaticity. The associated value is a floating-point value packaged as an `NSNumber` object. You can query this value to get the current x value for neutral x, y.

  Available in Mac OS X v10.5 and later.

  Declared in `CIRAWFilter.h`.

`kCIInputNeutralChromaticityYKey`
: The y value of the chromaticity. The associated value is a floating-point value packaged as an `NSNumber` object. You can query this value to get the current y value for neutral x, y.

  Available in Mac OS X v10.5 and later.

  Declared in `CIRAWFilter.h`.

`kCIInputNeutralTemperatureKey`
: A key for neutral temperature. The associated value is a floating-point value packaged as an `NSNumber` object. You can query this value to get the current temperature value.

  Available in Mac OS X v10.5 and later.

  Declared in `CIRAWFilter.h`.

`kCIInputNeutralTintKey`
: A key for the neutral tint. The associated value is a floating-point value packaged as an `NSNumber` object. Use this key to set or fetch the temperature and tint values. You can query this value to get the current tint value.

  Available in Mac OS X v10.5 and later.

  Declared in `CIRAWFilter.h`.

`kCIInputNeutralLocationKey`
: A key for the neutral position. Use this key to set the location in geometric coordinates of the unrotated output image that should be used as neutral. You cannot query this value; it is undefined for reading. The associated value is a two-element `CIVector` object that specifies the location (`x`, `y`).

  Available in Mac OS X v10.5 and later.

  Declared in `CIRAWFilter.h`.

`kCIInputScaleFactorKey`
: A key for the scale factor. The associated value is a floating-point value packaged as an `NSNumber` object that specifies the desired scale factor at which the image will be drawn. Setting this value can greatly improve the drawing performance. A value of `1` is the identity. In some cases, if you change the scale factor and enable draft mode, performance can decrease. See `kCIAllowDraftModeKey`.

  Available in Mac OS X v10.5 and later.

  Declared in `CIRAWFilter.h`.

`kCIInputAllowDraftModeKey`
: A key for allowing draft mode. The associated value is a Boolean value packaged as an `NSNumber` object. It's best not to use draft mode if the image needs to be drawn without draft mode at a later time, because changing the value from `YES` to `NO` is an expensive operation. If the optional scale factor is smaller than a certain value, additionally setting draft mode can improve image decoding speed without any perceivable loss of quality. However, turning on draft mode does not have any effect if the scale factor is not below this threshold.

  Available in Mac OS X v10.5 and later.

  Declared in `CIRAWFilter.h`.

`kCIInputIgnoreImageOrientationKey`
: A key for specifying whether to ignore the image orientation. The associated value is a Boolean value packaged as an `NSNumber` object. The default value is `NO`. An image is usually loaded in its proper orientation, as long as the associated metadata records its orientation. For special purposes you might want to load the image in its physical orientation. The exact meaning of "physical orientation" is dependent on the specific image.

  Available in Mac OS X v10.5 and later.

  Declared in `CIRAWFilter.h`.

`kCIInputImageOrientationKey`
: A key for the image orientation. The associated value is an integer value packaged as an `NSNumber` object. Valid values are in range `1...8` and follow the EXIF specification. The value is disregarded when the `kCIIgnoreImageOrientationKey` flag is set. You can change the orientation of the image by overriding this value. By changing this value you can easily rotate an image in 90-degree increments.

  Available in Mac OS X v10.5 and later.

  Declared in `CIRAWFilter.h`.

`kCIInputEnableSharpeningKey`
: A key for the sharpening state. The associated value must be an `NSNumber` object that specifies a `BOOL` value (`YES` or `NO`). The default is `YES`. This option has no effect if the image used for initialization is not RAW.

  Available in Mac OS X v10.5 and later.

  Declared in `CIRAWFilter.h`.

`kCIInputEnableChromaticNoiseTrackingKey`
: A key for progressive chromatic noise tracking (based on ISO and exposure time). The associated value must be an `NSNumber` object that specifies a `BOOL` value (`YES` or `NO`). The default is `YES`. This option has no effect if the image used for initialization is not RAW.

  Available in Mac OS X v10.5 and later.

  Declared in `CIRAWFilter.h`.

`kCIInputBoostShadowAmountKey`
: A key for the amount to boost the shadow areas of the image. The associated value must be an `NSNumber` object that specifies floating-point value. The value has no effect if the image used for initialization is not RAW.

  Available in Mac OS X v10.5 and later.

  Declared in `CIRAWFilter.h`.

`kCIInputBiasKey`
: A key for the simple bias value to use along with the exposure adjustment (`kCIInputEVKey`). The associated value must be an `NSNumber` object that specifies floating-point value. The value has no effect if the image used for initialization is not RAW.

  Available in Mac OS X v10.5 and later.

  Declared in `CIRAWFilter.h`.

Discussion
You can also use the key `kCIInputEVKey` for RAW images.

Declared In
`CIRAWFilter.h`

CIFilter Core Animation Additions

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/QuartzCore.framework
Declared in	CACIFilterAdditions.h
Companion guides	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 182) *property*
The name of the receiver.

Enabling Filter Instances

`enabled` (page 182) *property*
Determines if the receiver is enabled. Animatable.

`– isEnabled` (page 182)
A synthesized accessor for the `enabled` (page 182) property.

Properties

For more about Objective-C properties, see “Properties” in *The Objective-C 2.0 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 182) property.

```
- (BOOL)isEnabled
```

See Also
`@property enabled` (page 182)

CIFilterGenerator Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	Library/Frameworks/QuartzCore.framework
Declared in	QuartzCore/CIFilterGenerator.h
Availability	Mac OS X v10.5 and later
Companion guides	Core Image Programming Guide Core Image Filter Reference

Overview

The `CIFilterGenerator` class provides methods for creating a `CIFilter` object by chaining together existing `CIFilter` objects to create complex effects. (A **filter chain** refers to the `CIFilter` objects that are connected in the `CIFilterGenerator` object.) The complex effect can be encapsulated as a `CIFilterGenerator` object and saved as a file so that it can be used again. The **filter generator file** contains an archived instance of all the `CIFilter` objects that are chained together.

Any filter generator files that you copy to `/Library/Graphics/Image Units/` are loaded when any of the loading methods provided by the `CIPlugIn` class are invoked. A `CIFilterGenerator` object is registered by its filename or, if present, by a class attribute that you supply in its description.

You can create a `CIFilterGenerator` object programmatically, using the methods provided by the `CIFilterGenerator` class, or by using the editor view provided by Core Image (see *CIFilter Image Kit Additions*).

Tasks

Creating Filter Generator Objects

`+ filterGenerator` (page 185)
Creates and returns an empty filter generator object.

`+ filterGeneratorWithContentsOfURL:` (page 185)
: Creates and returns a filter generator object and initializes it with the contents of a filter generator file.

Initializing a Filter Generator Object

`- initWithContentsOfURL:` (page 189)
: Initializes a filter generator object with the contents of a filter generator file.

Connecting and Disconnecting Objects

`- connectObject:withKey:toObject:withKey:` (page 186)
: Adds an object to the filter chain.

`- disconnectObject:withKey:toObject:withKey:` (page 187)
: Removes the connection between two objects in the filter chain.

Managing Exported Keys

`- exportedKeys` (page 187)
: Returns an array of the exported keys.

`- exportKey:fromObject:withName:` (page 188)
: Exports an input or output key of an object in the filter chain.

`- removeExportedKey:` (page 190)
: Removes a key that was previously exported.

`- setAttributes:forExportedKey:` (page 190)
: Sets a dictionary of attributes for an exported key.

Setting and Getting Class Attributes

`- classAttributes` (page 186)
: Retrieves the class attributes associated with a filter.

`- setClassAttributes:` (page 190)
: Seta the class attributes for a filter.

Archiving a Filter Generator Object

`- writeToURL:atomically:` (page 191)
: Archives a filter generator object to a filter generator file.

Registering a Filter Chain

`- registerFilterName:` (page 189)
Registers the name associated with a filter chain.

Creating a Filter from a Filter Chain

`- filter` (page 188)
Creates a filter object based on the filter chain.

Class Methods

filterGenerator

Creates and returns an empty filter generator object.

```
+ (CIFilterGenerator *)filterGenerator
```

Return Value
A `CIFilterGenerator` object.

Discussion
You use the returned object to connect two or more `CIFilter` objects and input images. It is also valid to have only one `CIFilter` object in a filter generator.

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

See Also
`+ filterGeneratorWithContentsOfURL:` (page 185)

Declared In
`CIFilterGenerator.h`

filterGeneratorWithContentsOfURL:

Creates and returns a filter generator object and initializes it with the contents of a filter generator file.

```
+ (CIFilterGenerator *)filterGeneratorWithContentsOfURL:(NSURL *)aURL
```

Parameters
aURL
The location of a filter generator file.

Return Value
A `CIFilterGenerator` object; returns `nil` if the file can't be read.

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

See Also
`+ filterGenerator` (page 185)

Declared In
`CIFilterGenerator.h`

Instance Methods

classAttributes

Retrieves the class attributes associated with a filter.

```
- (NSDictionary *)classAttributes
```

Return Value
An `NSDictionary` object that contains the class attributes for a filter, or `nil` if attributes are not set for the filter.

Discussion
For more information about class attributes for a filter, see *Core Image Programming Guide* and the filter attributes key constants defined in *CIFilter Class Reference*.

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

See Also
`- setClassAttributes:` (page 190)

Declared In
`CIFilterGenerator.h`

connectObject:withKey:toObject:withKey:

Adds an object to the filter chain.

```
- (void)connectObject:(id)sourceObject withKey:(NSString *)sourceKey
    toObject:(id)targetObject withKey:(NSString *)targetKey
```

Parameters

sourceObject
A `CIFilter` object, a `CIImage` object, or a the path (an `NSString` or `NSURL` object) to an image.

sourceKey
The key that specifies the source object. For example, if the source is the output image of a filter, pass the `outputImage` key. Pass `nil` if the source object is used directly.

targetObject
The object that to link the source object to.

targetKey
The key that specifies the target for the source. For example, if you are connecting the source to the input image of a `CIFilter` object, you would pass the `inputImage` key.

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

See Also
`- disconnectObject:withKey:toObject:withKey:` (page 187)

Declared In
`CIFilterGenerator.h`

disconnectObject:withKey:toObject:withKey:

Removes the connection between two objects in the filter chain.

```
- (void)disconnectObject:(id)sourceObject withKey:(NSString *)key
    toObject:(id)targetObject withKey:(NSString *)targetKey
```

Parameters

sourceObject
A `CIFilter` object, a `CIImage` object, or a the path (an `NSString` or `NSURL` object) to an image.

sourceKey
The key that specifies the source object. Pass `nil` if the source object is used directly.

targetObject
The object that you want to disconnect the source object from.

targetKey
The key that specifies the target that the source object is currently connected to.

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

See Also
`- connectObject:withKey:toObject:withKey:` (page 186)

Declared In
`CIFilterGenerator.h`

exportedKeys

Returns an array of the exported keys.

```
- (NSDictionary *)exportedKeys
```

Return Value
An array of dictionaries that describe the exported key and target object. See `kCIFilterGeneratorExportedKey` (page 192), `kCIFilterGeneratorExportedKeyTargetObject` (page 192), and `kCIFilterGeneratorExportedKey` (page 192) for keys used in the dictionary.

Discussion
This method returns the keys that you exported using the `exportKey:fromObject:withName:` (page 188) method or that were exported before being written to the file from which you read the filter chain.

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

See Also

- `exportKey:fromObject:withName:` (page 188)

Declared In

`CIFilterGenerator.h`

exportKey:fromObject:withName:

Exports an input or output key of an object in the filter chain.

```
- (void)exportKey:(NSString *)key fromObject:(id)targetObject withName:(NSString
    *)exportedKeyName
```

Parameters

key

The key to export from the target object (for example, `inputImage`).

targetObject

The object associated with the key (for example, the filter).

exportedKeyName

A unique name to use for the exported key. Pass `nil` to use the original key name.

Discussion

When you create a `CIFilter` object from a `CIFilterGenerator` object, you might want the filter client to be able to set some of the parameters associated with the filter chain. You can make a parameter settable by exporting the key associated with the parameter. If the exported key represents an input parameter of the filter, the key is exported as an input key. If the key represents an output parameter, it is exported as an output key.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `exportedKeys` (page 187)
- `setAttributes:forExportedKey:` (page 190)
- `removeExportedKey:` (page 190)

Declared In

`CIFilterGenerator.h`

filter

Creates a filter object based on the filter chain.

```
- (CIFilter *)filter
```

Return Value

A `CIFilter` object.

Discussion

The topology of the filter chain is immutable, meaning that any changes you make to the filter chain are not reflected in the filter. The returned filer has the input an output keys that are exported.

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

Declared In
`CIFilterGenerator.h`

initWithContentsOfURL:

Initializes a filter generator object with the contents of a filter generator file.

```
- (id)initWithContentsOfURL:(NSURL *)aURL
```

Parameters

aURL
    The location of a filter generator file.

Return Value
The initialized `CIFilterGenerator` object. Returns `nil` if the file can't be read.

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

See Also
`+ filterGenerator` (page 185)
`+ filterGeneratorWithContentsOfURL:` (page 185)

Declared In
`CIFilterGenerator.h`

registerFilterName:

Registers the name associated with a filter chain.

```
- (void)registerFilterName:(NSString *)name
```

Parameters

name
    A unique name for the filter chain you want to register.

Discussion
This method allows you to register the filter chain as a named filter in the Core Image filter repository. You can then create a `CIFilter` object from it using the the `filterWithName:` (page 157) method of the `CIFilter` class.

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

Declared In
`CIFilterGenerator.h`

removeExportedKey:

Removes a key that was previously exported.

```
- (void)removeExportedKey:(NSString *)exportedKeyName
```

Parameters

exportedKeyName

The name of the key you want to remove.

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

See Also
`– exportKey:fromObject:withName:` (page 188)

Declared In
`CIFilterGenerator.h`

setAttributes:forExportedKey:

Sets a dictionary of attributes for an exported key.

```
- (void)setAttributes:(NSDictionary *)attributes forExportedKey:(NSString *)key
```

Parameters

attributes

A dictionary that describes the attributes associated with the specified key.

key

The exported key whose attributes you want to set.

Discussion
By default, the exported key inherits the attributes from its original key and target object. You can use this method to change one or more of the existing attributes for the key, such as the default value or maximum value. For more information on attributes, see *CIFilter Class Reference* and *Core Image Programming Guide*.

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

See Also
`– exportedKeys` (page 187)
`– exportKey:fromObject:withName:` (page 188)

Declared In
`CIFilterGenerator.h`

setClassAttributes:

Seta the class attributes for a filter.

```
- (void)setClassAttributes:(NSDictionary *)attributes
```

Parameters

attributes

An `NSDictionary` object that contains the class attributes for a filter For information on the required attributes, see *CIFilter Class Reference* and *Core Image Programming Guide*.

Availability

Available in Mac OS X v10.5 and later.

See Also

`– classAttributes` (page 186)

Declared In

`CIFilterGenerator.h`

writeToURL:atomically:

Archives a filter generator object to a filter generator file.

```
- (BOOL)writeToURL:(NSURL *)aURL atomically:(BOOL)flag
```

Parameters

aURL

A location for the file generator file.

flag

Pass `true` to specify that Core Image should create an interim file to avoid overwriting an existing file.

Return Value

Returns `true` if the the object is successfully archived to the file.

Discussion

Use this method to save your filter chain to a file for later use.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CIFilterGenerator.h`

Constants

Exported Keys

Keys for the exported parameters of a filter generator object.

```
extern NSString *const kCIFilterGeneratorExportedKey;
extern NSString *const kCIFilterGeneratorExportedKeyTargetObject;
extern NSString *const kCIFilterGeneratorExportedKeyName;
```

Constants

`kCIFilterGeneratorExportedKeyName`

The key (`CIFilterGeneratorExportedKeyName`) for the name used to export the `CIFilterGenerator` object. The associated value is a string that specifies a unique name for the filter generator object.

Available in Mac OS X v10.5 and later.

Declared in `CIFilterGenerator.h`.

`kCIFilterGeneratorExportedKey`

The key (`CIFilterGeneratorExportedKey`) for the exported parameter. The associated value is the key name of the parameter you are exporting, such as `inputRadius`.

Available in Mac OS X v10.5 and later.

Declared in `CIFilterGenerator.h`.

`kCIFilterGeneratorExportedKeyTargetObject`

The target object (`CIFilterGeneratorExportedKeyTargetObject`) for the exported key. The associated value is the name of the object, such as `CIMotionBlur`.

Available in Mac OS X v10.5 and later.

Declared in `CIFilterGenerator.h`.

Declared In

`CIFilterGenerator.h`

CIFilterShape Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSObject (NSObject)
Framework	Library/Frameworks/QuartzCore.framework
Declared in	QuartzCore/CIFilterShape.h
Availability	Mac OS X v10.4 and later
Companion guide	Core Image Programming Guide
Related sample code	CIAnnotation

Overview

The `CIFilterShape` class describes the bounding shape of a filter and the domain of definition (DOD) of a filter operation. You use `CIFilterShape` objects in conjunction with Core Image classes, such as `CIFilter`, `CIKernel`, and `CISampler`, to create custom filters.

Tasks

Creating a Filter Shape

`+ shapeWithRect:` (page 194)
Creates a filter shape object and initializes it with a rectangle.

Initializing a Filter Shape

`- initWithRect:` (page 195)
Initializes a filter shape object with a rectangle.

Modifying a Filter Shape

`- insetByX:Y:` (page 195)
    Modifies a filter shape object so that it is inset by the specified x and y values.

`- intersectWith:` (page 195)
    Creates a filter shape object that represents the intersection of the current filter shape and the specified filter shape object.

`- intersectWithRect:` (page 196)
    Creates a filter shape that represents the intersection of the current filter shape and a rectangle.

`- transformBy:interior:` (page 196)
    Creates a filter shape that results from applying a transform to the current filter shape.

`- unionWith:` (page 197)
    Creates a filter shape that results from the union of the current filter shape and another filter shape object.

`- unionWithRect:` (page 197)
    Creates a filter shape that results from the union of the current filter shape and a rectangle.

Class Methods

shapeWithRect:

Creates a filter shape object and initializes it with a rectangle.

```
+ (id)shapeWithRect:(CGRect)r
```

Parameters

r
    A rectangle. The filter shape object will contain the smallest integral rectangle specified by this argument.

Availability
Mac OS X v10.4 and later.

See Also
`- initWithRect:` (page 195)

Related Sample Code
CIAnnotation

Declared In
`CIFilterShape.h`

Instance Methods

initWithRect:

Initializes a filter shape object with a rectangle.

```
- (id)initWithRect:(CGRect)r
```

Parameters

r

A rectangle. Core Image uses the rectangle specified by integer parts of the values in the `CGRect` data structure.

Return Value

An initialized CIFilterShape object, or `nil` if the method fails.

Availability

Mac OS X v10.4 and later.

See Also

`+ shapeWithRect:` (page 194)

Declared In

`CIFilterShape.h`

insetByX:Y:

Modifies a filter shape object so that it is inset by the specified x and y values.

```
- (CIFilterShape *)insetByX:(int)dx Y:(int)dy
```

Parameters

dx

A value that specifies an inset in the x direction.

dy

A value that specifies an inset in the y direction.

Availability

Mac OS X v10.4 and later.

Declared In

`CIFilterShape.h`

intersectWith:

Creates a filter shape object that represents the intersection of the current filter shape and the specified filter shape object.

```
- (CIFilterShape *)intersectWith:(CIFilterShape *)s2
```

Parameters

s2

A filter shape object.

Return Value

The filter shape object that results from the intersection.

Availability

Mac OS X v10.4 and later.

See Also

`- intersectWithRect:` (page 196)

Declared In

`CIFilterShape.h`

intersectWithRect:

Creates a filter shape that represents the intersection of the current filter shape and a rectangle.

```
- (CIFilterShape *)intersectWithRect:(CGRect)r
```

Parameters

rect

A rectangle. Core Image uses the rectangle specified by integer parts of the width and height.

Return Value

The filter shape that results from the intersection

Availability

Mac OS X v10.4 and later.

See Also

`- intersectWith:` (page 195)

Declared In

`CIFilterShape.h`

transformBy:interior:

Creates a filter shape that results from applying a transform to the current filter shape.

```
- (CIFilterShape *)transformBy:(CGAffineTransform)m interior:(BOOL)flag
```

Parameters

m

A transform.

flag

`NO` specifies that the new filter shape object can contain all the pixels in the transformed shape (and possibly some that are outside the transformed shape). `YES` specifies that the new filter shape object can contain a subset of the pixels in the transformed shape (but none of those outside the transformed shape).

Return Value
The transformed filter shape object.

Availability
Mac OS X v10.4 and later.

Declared In
`CIFilterShape.h`

unionWith:

Creates a filter shape that results from the union of the current filter shape and another filter shape object.

```
- (CIFilterShape *)unionWith:(CIFilterShape *)s2
```

Parameters

s2
A filter shape object.

Return Value
The filter shape object that results from the union.

Availability
Mac OS X v10.4 and later.

See Also
`- unionWithRect:` (page 197)

Declared In
`CIFilterShape.h`

unionWithRect:

Creates a filter shape that results from the union of the current filter shape and a rectangle.

```
- (CIFilterShape *)unionWithRect:(CGRect)r
```

Parameters

rect
A rectangle. Core Image uses the rectangle specified by integer parts of the width and height.

Availability
Mac OS X v10.4 and later.

See Also
`- unionWith:` (page 197)

Related Sample Code
CIAnnotation

Declared In
`CIFilterShape.h`

CIImage Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	Library/Frameworks/QuartzCore.framework
Declared in	QuartzCore/CIImage.h QuartzCore/CIImageProvider.h
Availability	Mac OS X v10.4 and later
Companion guide	Core Image Programming Guide
Related sample code	CarbonCocoaCoreImageTab CIAnnotation CITransitionSelectorSample2 Reducer WebKitCIPlugIn

Overview

The `CIImage` class represents an image. Core Image images are immutable. You use `CIImage` objects in conjunction with other Core Image classes, such as `CIFilter`, `CIContext`, `CIVector`, and `CIColor`, to take advantage of the built-in Core Image filters when processing images. You can create `CIImage` objects with data supplied from a variety of sources, including Quartz 2D images, Core Video image buffers (`CVImageBufferRef` (page 338)), URL-based objects, and `NSData` objects.

Although a `CIImage` object has image data associated with it, it is not an image. You can think of a `CIImage` object as an image "recipe." A `CIImage` object has all the information necessary to produce an image, but Core Image doesn't actually render an image until it is told to do so. This "lazy evaluation" method allows Core Image to operate as efficiently as possible.

Core Image defines methods for creating and initializing images. Additional methods that support drawing and initializing an image with an `NSBitmapImageRep` object are defined in *CIImage Additions Reference*.

Tasks

Creating an Image

+ `emptyImage` (page 202)
  Creates and returns an empty image object.

+ `imageWithColor:` (page 204)
  Creates and returns an image of infinite extent that is initialized the specified color.

+ `imageWithBitmapData:bytesPerRow:size:format:colorSpace:` (page 202)
  Creates and returns an image object from bitmap data.

+ `imageWithCGImage:` (page 203)
  Creates and returns an image object from a Quartz 2D image.

+ `imageWithCGImage:options:` (page 203)
  Creates and returns an image object from a Quartz 2D image using the specified color space.

+ `imageWithCGLayer:` (page 204)
  Creates and returns an image object from the contents supplied by a `CGLayer` object.

+ `imageWithCGLayer:options:` (page 204)
  Creates and returns an image object from the contents supplied by a `CGLayer` object, using the specified options.

+ `imageWithContentsOfURL:` (page 205)
  Creates and returns an image object from the contents of a file.

+ `imageWithContentsOfURL:options:` (page 205)
  Creates and returns an image object from the contents of a file, using the specified options.

+ `imageWithCVImageBuffer:` (page 206)
  Creates and returns an image object from the contents of `CVImageBuffer` object.

+ `imageWithCVImageBuffer:options:` (page 207)
  Creates and returns an image object from the contents of `CVImageBuffer` object, using the specified options.

+ `imageWithData:` (page 207)
  Creates and returns an image object initialized with the supplied image data.

+ `imageWithData:options:` (page 208)
  Creates and returns an image object initialized with the supplied image data, using the specified options.

+ `imageWithImageProvider:size:format:colorSpace:options:` (page 208)
  Creates and returns an image object initialized with data provided by an image provider.

+ `imageWithTexture:size:flipped:colorSpace:` (page 209)
  Creates and returns an image object initialized with data supplied by an OpenGL texture.

Creating an Image by Modifying an Existing Image

– `imageByApplyingTransform:` (page 211)
  Returns a new image that represents the original image after applying an affine transform.

- `imageByCroppingToRect:` (page 211)
  Returns a new image that represents the original image after cropping to a rectangle.

Initializing an Image

- `initWithColor:` (page 214)
  Initializes an image with the specified color.
- `initWithBitmapData:bytesPerRow:size:format:colorSpace:` (page 211)
  Initializes an image object with bitmap data.
- `initWithCGImage:` (page 212)
  Initializes an image object with a Quartz 2D image.
- `initWithCGImage:options:` (page 213)
  Initializes an image object with a Quartz 2D image, using the specified options.
- `initWithCGLayer:` (page 213)
  Initializes an image object from the contents supplied by a CGLayer object.
- `initWithCGLayer:options:` (page 214)
  Initializes an image object from the contents supplied by a CGLayer object, using the specified options.
- `initWithContentsOfURL:` (page 214)
  Initializes an image object from the contents of a file.
- `initWithContentsOfURL:options:` (page 215)
  Initializes an image object from the contents of a file, using the specified options.
- `initWithCVImageBuffer:` (page 215)
  Initializes an image object from the contents of CVImageBuffer object.
- `initWithCVImageBuffer:options:` (page 216)
  Initializes an image object from the contents of CVImageBuffer object, using the specified options.
- `initWithData:` (page 216)
  Initializes an image object with the supplied image data.
- `initWithData:options:` (page 217)
  Initializes an image object with the supplied image data, using the specified options.
- `initWithImageProvider:size:format:colorSpace:options:` (page 217)
  Initializes an image object with data provided by an image provider, using the specified options.
- `initWithTexture:size:flipped:colorSpace:` (page 218)
  Initializes an image object with data supplied by an OpenGL texture.

Getting Image Information

- `definition` (page 210)
  Returns a filter shape object that represents the domain of definition of the image.
- `extent` (page 210)
  Returns a rectangle that specifies the extent of the image.

Class Methods

emptyImage

Creates and returns an empty image object.

```
+ (CIImage *)emptyImage
```

Return Value
An image object.

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

Declared In
`CIImage.h`

imageWithBitmapData:bytesPerRow:size:format:colorSpace:

Creates and returns an image object from bitmap data.

```
+ (CIImage *)imageWithBitmapData:(NSData *)d bytesPerRow:(size_t)bpr
    size:(CGSize)size format:(CIFormat)f colorSpace:(CGColorSpaceRef)cs
```

Parameters

d
: The bitmap data for the image. This data must be premultiplied.

bpr
: The number of bytes per row.

size
: The dimensions of the image.

f
: The format and size of each pixel. You must supply a pixel format constant. See "`Pixel Formats`" (page 219).

cs
: The color space that the image is defined in. If this value is `nil`, the image is not color matched. Pass `nil` for images that don't contain color data (such as elevation maps, normal vector maps, and sampled function tables).

Return Value
An image object.

Availability
Mac OS X v10.4 and later.

See Also
`- initWithBitmapData:bytesPerRow:size:format:colorSpace:` (page 211)

Declared In
`CIImage.h`

imageWithCGImage:

Creates and returns an image object from a Quartz 2D image.

```
+ (CIImage *)imageWithCGImage:(CGImageRef)image
```

Parameters

image

A Quartz 2D image (`CGImageRef`) object. For more information, see *Quartz 2D Programming Guide* and *CGImage Reference*.

Return Value

An image object initialized with the contents of the Quartz 2D image.

Availability

Mac OS X v10.4 and later.

See Also

`+ imageWithCGImage:options:` (page 203)
`- initWithCGImage:` (page 212)

Related Sample Code

CIVideoDemoGL

Declared In

`CIImage.h`

imageWithCGImage:options:

Creates and returns an image object from a Quartz 2D image using the specified color space.

```
+ (CIImage *)imageWithCGImage:(CGImageRef)image options:(NSDictionary *)d
```

Parameters

image

A Quartz 2D image (`CGImageRef`) object. For more information, see *Quartz 2D Programming Guide* and *CGImage Reference*.

d

A dictionary that contains a color space key (`kCIImageColorSpace` (page 220)) whose value is a `CGColorSpace`object. (See `CGColorSpaceRef`.)

Return Value

An image object initialized with the contents of the Quartz 2D image and the specified color space.

Availability

Mac OS X v10.4 and later.

See Also

`+ imageWithCGImage:` (page 203)
`- initWithCGImage:options:` (page 213)

Declared In

`CIImage.h`

imageWithCGLayer:

Creates and returns an image object from the contents supplied by a `CGLayer` object.

```
+ (CIImage *)imageWithCGLayer:(CGLayerRef)layer
```

Parameters

layer

A `CGLayer` object. For more information see *Quartz 2D Programming Guide* and *CGLayer Reference*.

Return Value

An image object initialized with the contents of the layer object.

Availability

Mac OS X v10.4 and later.

See Also

`+ imageWithCGLayer:options:` (page 204)

`- initWithCGLayer:` (page 213)

Declared In

`CIImage.h`

imageWithCGLayer:options:

Creates and returns an image object from the contents supplied by a `CGLayer` object, using the specified options.

```
+ (CIImage *)imageWithCGLayer:(CGLayerRef)layer options:(NSDictionary *)d
```

Parameters

layer

A `CGLayer` object. For more information see *Quartz 2D Programming Guide* and *CGLayer Reference*.

d

A dictionary that contains options for creating an image object. You can supply such options as a pixel format and a color space. See "Pixel Formats" (page 219).

Return Value

An image object initialized with the contents of the layer object and set up with the specified options.

Availability

Mac OS X v10.4 and later.

See Also

`+ imageWithCGLayer:` (page 204)

`- initWithCGLayer:options:` (page 214)

Declared In

`CIImage.h`

imageWithColor:

Creates and returns an image of infinite extent that is initialized the specified color.

```
+ (CIImage *)imageWithColor:(CIColor *)color
```

Parameters

color

A color object.

Return Value

The image object initialized with the color represented by the `CIColor` object.

Availability

Available in Mac OS X v10.5 and later.

See Also

`- initWithColor:` (page 214)

Declared In

`CIImage.h`

imageWithContentsOfURL:

Creates and returns an image object from the contents of a file.

```
+ (CIImage *)imageWithContentsOfURL:(NSURL *)url
```

Parameters

url

The location of the file.

Return Value

An image object initialized with the contents of the file.

Availability

Mac OS X v10.4 and later.

See Also

`+ imageWithContentsOfURL:options:` (page 205)
`- initWithContentsOfURL:` (page 214)

Related Sample Code

CIAnnotation
CITransitionSelectorSample2

Declared In

`CIImage.h`

imageWithContentsOfURL:options:

Creates and returns an image object from the contents of a file, using the specified options.

```
+ (CIImage *)imageWithContentsOfURL:(NSURL *)url options:(NSDictionary *)d
```

Parameters

`url`

The location of the file.

`d`

A dictionary that contains options for creating an image object. You can supply such options as a pixel format and a color space. See "Pixel Formats" (page 219).

Return Value

An image object initialized with the contents of the file and set up with the specified options.

Availability

Mac OS X v10.4 and later.

See Also

`+ imageWithContentsOfURL:` (page 205)

`- initWithContentsOfURL:options:` (page 215)

Declared In

`CIImage.h`

imageWithCVImageBuffer:

Creates and returns an image object from the contents of `CVImageBuffer` object.

```
+ (CIImage *)imageWithCVImageBuffer:(CVImageBufferRef)imageBuffer
```

Parameters

`imageBuffer`

A `CVImageBuffer` object. For more information, see *Core Video Programming Guide* and *Core Video Reference*.

Return Value

An image object initialized with the contents of the image buffer object.

Availability

Mac OS X v10.4 and later.

See Also

`+ imageWithCVImageBuffer:options:` (page 207)

`- initWithCVImageBuffer:` (page 215)

Related Sample Code

CIVideoDemoGL

QTCarbonCoreImage101

QTCoreImage101

WhackedTV

Declared In

`CIImage.h`

imageWithCVImageBuffer:options:

Creates and returns an image object from the contents of `CVImageBuffer` object, using the specified options.

```
+ (CIImage *)imageWithCVImageBuffer:(CVImageBufferRef)imageBuffer
    options:(NSDictionary *)dict
```

Parameters

imageBuffer

A `CVImageBuffer` object. For more information, see *Core Video Programming Guide* and *Core Video Reference*.

dict

A dictionary that contains options for creating an image object. You can supply such options as a color space. (The pixel format is supplied by the `CVImageBuffer` object.)

Return Value
An image object initialized with the contents of the image buffer object and set up with the specified options.

Availability
Mac OS X v10.4 and later.

See Also
`+ imageWithCVImageBuffer:` (page 206)
`- initWithCVImageBuffer:options:` (page 216)

Declared In
`CIImage.h`

imageWithData:

Creates and returns an image object initialized with the supplied image data.

```
+ (CIImage *)imageWithData:(NSData *)data
```

Parameters

data

The data object that holds the contents of an image file (such as TIFF, GIF, JPG, or whatever else the system supports). The image data must be premultiplied.

Return Value
An image object initialized with the supplied data, or `nil` if the method cannot create an image representation from the contents of the supplied data object.

Availability
Mac OS X v10.4 and later.

See Also
`+ imageWithData:options:` (page 208)
`- initWithData:` (page 216)

Related Sample Code
LayerBackedOpenGLView
WebKitCIPlugIn

Declared In
`CIImage.h`

imageWithData:options:

Creates and returns an image object initialized with the supplied image data, using the specified options.

```
+ (CIImage *)imageWithData:(NSData *)data options:(NSDictionary *)d
```

Parameters

data

A pointer to the image data. The data must be premultiplied

d

A dictionary that contains options for creating an image object. You can supply such options as a pixel format and a color space. See "`Pixel Formats`" (page 219).

Return Value
An image object initialized with the supplied data and set up with the specified options.

Availability
Mac OS X v10.4 and later.

See Also
`+ imageWithData:` (page 207)
`- initWithData:options:` (page 217)

Declared In
`CIImage.h`

imageWithImageProvider:size:format:colorSpace:options:

Creates and returns an image object initialized with data provided by an image provider.

```
+ (CIImage *)imageWithImageProvider:(id)p size:(size_t)width :(size_t)height
    format(CIFormat)f colorSpace:(CGColorSpaceRef)cs options:(NSDictionary *)dict
```

Parameters

p

A data provider that implements the `CIImageProvider` informal protocol. Core Image retains this data until the image is deallocated.

width

The width of the image.

height

The height of the image.

f

A pixel format constant. See "`Pixel Formats`" (page 219).

cs

The color space that the image is defined in. If the this value is `nil`, the image is not color matched. Pass `nil` for images that don't contain color data (such as elevation maps, normal vector maps, and sampled function tables).

dict

A dictionary that specifies image-creation options, which can be `kCIImageProviderTileSize` or `kCIImageProviderUserInfo`. See *CIImageProvider Protocol Reference* for more information on these options.

Return Value

An image object initialized with the data from the data provider. Core Image does not populate the image object until the object needs the data.

Availability

Mac OS X v10.4 and later.

Declared In

`CIImageProvider.h`

See Also

`- initWithImageProvider:size::format:colorSpace:options:` (page 217)

imageWithTexture:size:flipped:colorSpace:

Creates and returns an image object initialized with data supplied by an OpenGL texture.

```
+ (CIImage *)imageWithTexture:(unsigned int)name size:(CGSize)size flipped:(BOOL)flag
    colorSpace:(CGColorSpaceRef)cs
```

Parameters

name

An OpenGL texture. Because `CIImage` objects are immutable, the texture must remain unchanged for the life of the image object. See the discussion for more information.

size

The dimensions of the texture.

flag

`YES` to have Core Image flip the contents of the texture vertically.

cs

The color space that the image is defined in. If the `colorSpace` value is `nil`, the image is not color matched. Pass `nil` for images that don't contain color data (such as elevation maps, normal vector maps, and sampled function tables).

Return Value

An image object initialized with the texture data.

Discussion

When using a texture to create a `CIImage` object, the texture must be valid in the Core Image context (`CIContext`) that you draw the `CIImage` object into. This means that one of the following must be true:

- The texture must be created using the `CGLContext` object that the `CIContext` is based on.
- The context that the texture was created in must be shared with the `CGLContext` that the `CIContext` is based on.

Note that textures do not have a retain and release mechanism. This means that your application must make sure that the texture exists for the life cycle of the image. When you no longer need the image, you can delete the texture.

Core Image ignores the texture filtering and wrap modes (`GL_TEXTURE_FILTER` and `GL_TEXTURE_WRAP`) that you set through OpenGL. The filter and wrap modes are overridden by what the CISampler object specifies when you apply a filter to the `CIImage` object.

Availability
Mac OS X v10.4 and later.

See Also
`- initWithTexture:size:flipped:colorSpace:` (page 218)

Declared In
`CIImage.h`

Instance Methods

definition

Returns a filter shape object that represents the domain of definition of the image.

```
- (CIFilterShape *)definition
```

Return Value
A filter shape object.

Availability
Mac OS X v10.4 and later.

See Also
`- extent` (page 210)

Declared In
`CIImage.h`

extent

Returns a rectangle that specifies the extent of the image.

```
- (CGRect)extent
```

Return Value
A rectangle that specifies the extent of the image in working space coordinates.

Availability
Mac OS X v10.4 and later.

See Also
`- definition` (page 210)

Related Sample Code
CIVideoDemoGL
QTCarbonCoreImage101

QTCoreImage101
Reducer
UnsharpMask

Declared In
`CIImage.h`

imageByApplyingTransform:

Returns a new image that represents the original image after applying an affine transform.

```
- (CIImage *)imageByApplyingTransform:(CGAffineTransform)matrix
```

Parameters

`matrix`
An affine transform.

Return Value
The transformed image object.

Availability
Mac OS X v10.4 and later.

See Also
`– imageByCroppingToRect:` (page 211)

Declared In
`CIImage.h`

imageByCroppingToRect:

Returns a new image that represents the original image after cropping to a rectangle.

```
- (CIImage *)imageByCroppingToRect:(CGRect)r
```

Return Value
An image object cropped to the specified rectangle.

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

See Also
`– imageByApplyingTransform:` (page 211)

Declared In
`CIImage.h`

initWithBitmapData:bytesPerRow:size:format:colorSpace:

Initializes an image object with bitmap data.

```
- (id)initWithBitmapData:(NSData *)d bytesPerRow:(size_t)bpr size:(CGSize)size
    format:(CIFormat)f colorSpace:(CGColorSpaceRef)c
```

Parameters

d

The bitmap data to use for the image. The data you supply must be premultiplied.

bpr

The number of bytes per row.

size

The size of the image data.

f

A pixel format constant. See "`Pixel Formats`" (page 219).

c

The color space that the image is defined in and must be a Quartz 2D color space (`CGColorSpaceRef`). Pass `nil` for images that don't contain color data (such as elevation maps, normal vector maps, and sampled function tables).

Return Value
The initialized image object or `nil` if the object could not be initialized.

Availability
Mac OS X v10.4 and later.

See Also
`+ imageWithBitmapData:bytesPerRow:size:format:colorSpace:` (page 202)

Declared In
`CIImage.h`

initWithCGImage:

Initializes an image object with a Quartz 2D image.

```
- (id)initWithCGImage:(CGImageRef)image
```

Parameters

image

A Quartz 2D image (`CGImageRef`) object. For more information, see *Quartz 2D Programming Guide* and *CGImage Reference*.

Return Value
The initialized image object or `nil` if the object could not be initialized.

Availability
Mac OS X v10.4 and later.

See Also
`- initWithCGImage:options:` (page 213)
`+ imageWithCGImage:` (page 203)

Declared In
`CIImage.h`

initWithCGImage:options:

Initializes an image object with a Quartz 2D image, using the specified options.

```
- (id)initWithCGImage:(CGImageRef)image options:(NSDictionary *)d
```

Parameters

image

A Quartz 2D image (`CGImageRef`) object. For more information, see *Quartz 2D Programming Guide* and *CGImage Reference*.

d

A dictionary that contains options for creating an image object. You can supply such options as a pixel format and a color space. See “`Pixel Formats`” (page 219).

Return Value
The initialized image object or `nil` if the object could not be initialized.

Availability
Mac OS X v10.4 and later.

See Also
`- initWithCGImage:` (page 212)
`+ imageWithCGImage:options:` (page 203)

Declared In
`CIImage.h`

initWithCGLayer:

Initializes an image object from the contents supplied by a CGLayer object.

```
- (id)initWithCGLayer:(CGLayerRef)layer
```

Parameters

layer

A CGLayer object. For more information see *Quartz 2D Programming Guide* and *CGLayer Reference*.

Return Value
The initialized image object or `nil` if the object could not be initialized.

Availability
Mac OS X v10.4 and later.

See Also
`- initWithCGLayer:options:` (page 214)
`+ imageWithCGLayer:` (page 204)

Related Sample Code
CIAnnotation
QTCarbonCoreImage101

Declared In
`CIImage.h`

initWithCGLayer:options:

Initializes an image object from the contents supplied by a CGLayer object, using the specified options.

```
- (id)initWithCGLayer:(CGLayerRef)layer options:(NSDictionary *)d
```

Parameters

layer

A CGLayer object. For more information see *Quartz 2D Programming Guide* and *CGLayer Reference*.

d

A dictionary that contains options for creating an image object. You can supply such options as a pixel format and a color space. See "`Pixel Formats`" (page 219).

Return Value

The initialized image object or `nil` if the object could not be initialized.

Availability

Mac OS X v10.4 and later.

See Also

`- initWithCGLayer:` (page 213)

`+ imageWithCGLayer:options:` (page 204)

Declared In

`CIImage.h`

initWithColor:

Initializes an image with the specified color.

```
- (id)initWithColor:(CIColor *)color
```

Parameters

color

A color object.

Return Value

The initialized image object or `nil` if the object could not be initialized.

Availability

Available in Mac OS X v10.5 and later.

See Also

`+ imageWithColor:` (page 204)

Declared In

`CIImage.h`

initWithContentsOfURL:

Initializes an image object from the contents of a file.

```
- (id)initWithContentsOfURL:(NSURL *)url
```

Parameters

url

The location of the file.

Return Value

The initialized image object or `nil` if the object could not be initialized.

Availability

Mac OS X v10.4 and later.

See Also

`- initWithContentsOfURL:options:` (page 215)

`+ imageWithContentsOfURL:` (page 205)

Declared In

`CIImage.h`

initWithContentsOfURL:options:

Initializes an image object from the contents of a file, using the specified options.

```
- (id)initWithContentsOfURL:(NSURL *)url options:(NSDictionary *)d
```

Parameters

url

The location of the file.

d

A dictionary that contains options for creating an image object. You can supply such options as a pixel format and a color space. See "`Pixel Formats`" (page 219).

Return Value

The initialized image object or `nil` if the object could not be initialized.

Availability

Mac OS X v10.4 and later.

See Also

`- initWithContentsOfURL:` (page 214)

`+ imageWithContentsOfURL:options:` (page 205)

Declared In

`CIImage.h`

initWithCVImageBuffer:

Initializes an image object from the contents of CVImageBuffer object.

```
- (id)initWithCVImageBuffer:(CVImageBufferRef)imageBuffer
```

Parameters

imageBuffer
: A CVImageBuffer object. For more information, see *Core Video Programming Guide* and *Core Video Reference*.

Return Value

The initialized image object or `nil` if the object could not be initialized.

Availability

Mac OS X v10.4 and later.

See Also

`- initWithCVImageBuffer:options:` (page 216)
`+ imageWithCVImageBuffer:` (page 206)

Related Sample Code

VideoViewer

Declared In

`CIImage.h`

initWithCVImageBuffer:options:

Initializes an image object from the contents of CVImageBuffer object, using the specified options.

```
- (id)initWithCVImageBuffer:(CVImageBufferRef)imageBuffer options:(NSDictionary
    *)dict
```

Parameters

imageBuffer
: A CVImageBuffer object. For more information, see *Core Video Programming Guide* and *Core Video Reference*.

dict
: A dictionary that contains options for creating an image object. You can supply such options as a color space. (The pixel format is supplied by the `CVImageBuffer` object.)

Return Value

The initialized image object or `nil` if the object could not be initialized.

Availability

Mac OS X v10.4 and later.

See Also

`- initWithCVImageBuffer:` (page 215)
`+ imageWithCVImageBuffer:options:` (page 207)

Declared In

`CIImage.h`

initWithData:

Initializes an image object with the supplied image data.

```
- (id)initWithData:(NSData *)data
```

Parameters

data

The image data. The data you supply must be premultiplied.

Return Value

The initialized image object or `nil` if the object could not be initialized.

Availability

Mac OS X v10.4 and later.

See Also

`- initWithData:options:` (page 217)

`+ imageWithData:` (page 207)

Declared In

`CIImage.h`

initWithData:options:

Initializes an image object with the supplied image data, using the specified options.

```
- (id)initWithData:(NSData *)data options:(NSDictionary *)d
```

Parameters

data

The image data. The data you supply must be premultiplied.

d

A dictionary that contains options for creating an image object. You can supply such options as a pixel format and a color space. See "`Pixel Formats`" (page 219).

Return Value

The initialized image object or `nil` if the object could not be initialized.

Availability

Mac OS X v10.4 and later.

See Also

`- initWithData:` (page 216)

`+ imageWithData:options:` (page 208)

Declared In

`CIImage.h`

initWithImageProvider:size:format:colorSpace:options:

Initializes an image object with data provided by an image provider, using the specified options.

```
- (id)initWithImageProvider:(id)p size:(size_t)width:(size_t)height
    format:(CIFormat)f colorSpace:(CGColorSpaceRef)cs options:(NSDictionary *)dict
```

Parameters

p

A data provider that implements the `CIImageProvider` informal protocol. Core Image retains this data until the image is deallocated.

width

The width of the image data.

height

The height of the image data.

f

A pixel format constant. See "`Pixel Formats`" (page 219).

cs

The color space of the image. If this value is `nil`, the image is not color matched. Pass `nil` for images that don't contain color data (such as elevation maps, normal vector maps, and sampled function tables).

dict

A dictionary that specifies image-creation options, which can be `kCIImageProviderTileSize` or `kCIImageProviderUserInfo`. See *CIImageProvider Protocol Reference* for more information on these options.

Return Value

The initialized image object or `nil` if the object could not be initialized.

Discussion

Core Image does not populate the image until it actually needs the data.

Availability

Mac OS X v10.4 and later.

Declared In

`CIImageProvider.h`

See Also

`+ imageWithImageProvider:size::format:colorSpace:options:` (page 208)

initWithTexture:size:flipped:colorSpace:

Initializes an image object with data supplied by an OpenGL texture.

```
- (id)initWithTexture:(unsigned int)name size:(CGSize)size flipped:(BOOL)flag
    colorSpace:(CGColorSpaceRef)cs
```

Parameters

name

An OpenGL texture. Because `CIImage` objects are immutable, the texture must remain unchanged for the life of the image object. See the discussion for more information.

size

The dimensions of the texture.

flag

`YES` to have Core Image flip the contents of the texture vertically.

`cs`
: The color space that the image is defined in. This must be a Quartz color space (`CGColorSpaceRef`). If the `colorSpace` value is `nil`, the image is not color matched. Pass `nil` for images that don't contain color data (such as elevation maps, normal vector maps, and sampled function tables).

Return Value
The initialized image object or `nil` if the object could not be initialized.

Discussion
When using a texture to create a `CIImage` object, the texture must be valid in the Core Image context (`CIContext`) that you draw the `CIImage` object into. This means that one of the following must be true:

- The texture must be created using the `CGLContext` object that the `CIContext` is based on.
- The context that the texture was created in must be shared with the `CGLContext` that the `CIContext`is based on.

Note that textures do not have a retain and release mechanism. This means that your application must make sure that the texture exists for the life cycle of the image. When you no longer need the image, you can delete the texture.

Core Image ignores the texture filtering and wrap modes (`GL_TEXTURE_FILTER` and `GL_TEXTURE_WRAP`) that you set through OpenGL. The filter and wrap modes are overridden by what the CISampler object specifies when you apply a filter to the `CIImage` object.

Availability
Mac OS X v10.4 and later.

See Also
`+ imageWithTexture:size:flipped:colorSpace:` (page 209)

Declared In
`CIImage.h`

Constants

Pixel Formats

Image data pixel formats.

```
extern CIFormat kCIFormatARGB8;
extern CIFormat kCIFormatRGBA16;
extern CIFormat kCIFormatRGBAf;
```

Constants

`CIFormat`
: The data type for a pixel format.

`kCIFormatARGB8`
: A 32 bit-per-pixel, fixed-point pixel format.

`kCIFormatRGBA16`
: A 64 bit-per-pixel, fixed-point pixel format.

`kCIFormatRGBAf`
A 128 bit-per-pixel, floating-point pixel format.

Declared In
`CIImage.h`

Color Space Key

A key for the color space of an image.

```
extern NSString *kCIImageColorSpace;
```

Constants

`kCIImageColorSpace`
The key for a color space. The value you supply for this dictionary key must be a `CGColorSpaceRef` data type. For more information on this data type see *CGColorSpace Reference*. Typically you use this option when you need to load an elevation, mask, normal vector, or RAW sensor data directly from a file without color correcting it. This constant specifies to override Core Image, which, by default, assumes that data is in GenericRGB.

Declared In
`CIImage.h`

CIImageAccumulator Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	Library/Frameworks/QuartzCore.framework
Declared in	QuartzCore/CIImageAccumulator.h
Availability	Mac OS X v10.4 and later
Companion guide	Core Image Programming Guide
Related sample code	CIAnnotation

Overview

The `CIImageAccumulator` class enables feedback-based image processing for such things as iterative painting operations or fluid dynamics simulations. You use `CIImageAccumulator` objects in conjunction with other Core Image classes, such as `CIFilter`, `CIImage`, `CIVector`, and `CIContext`, to take advantage of the built-in Core Image filters when processing images.

Tasks

Creating an Image Accumulator

`+ imageAccumulatorWithExtent:format:` (page 222)
Creates an image accumulator with the specified extent and pixel format.

Initializing an Image Accumulator

`- initWithExtent:format:` (page 224)
Initializes an image accumulator with the specified extent and pixel format.

Setting an Image

`- setImage:` (page 224)
    Sets the contents of the image accumulator to the contents of the specified image object.

`- setImage:dirtyRect:` (page 225)
    Updates an image accumulator with a subregion of an image object.

Obtaining Data From an Image Accumulator

`- extent` (page 223)
    Returns the extent of the image associated with the image accumulator.

`- format` (page 223)
    Returns the pixel format of the image accumulator.

`- image` (page 224)
    Returns the current contents of the image accumulator.

Resetting an Accumulator

`- clear` (page 223)
    Resets the accumulator, discarding any pending updates and the current content.

Class Methods

imageAccumulatorWithExtent:format:

Creates an image accumulator with the specified extent and pixel format.

```
+ (CIImageAccumulator *)imageAccumulatorWithExtent:(CGRect)r format:(CIFormat)f
```

Parameters

r
    A rectangle that specifies the x-value of the rectangle origin, the y-value of the rectangle origin, and the width and height.

f
    The format and size of each pixel. You must supply a pixel format constant, such as `kCIFormatARGB8` (32 bit-per-pixel, fixed-point pixel format) or `kCIFormatRGBAf` (128 bit-per-pixel, floating-point pixel format). See *CIImage Class Reference* for more information about pixel format constants.

Return Value
The image accumulator object.

Availability
Mac OS X v10.4 and later.

See Also
`- initWithExtent:format:` (page 224)

Declared In
`CIImageAccumulator.h`

Instance Methods

clear

Resets the accumulator, discarding any pending updates and the current content.

```
- (void)clear
```

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

Declared In
`CIImageAccumulator.h`

extent

Returns the extent of the image associated with the image accumulator.

```
- (CGRect)extent
```

Return Value
The rectangle that specifies the size of the image associated with the image accumulator. This rectangle is the size of the complete region of the working coordinate space, and is a fixed area. It specifies the x-value of the rectangle origin, the y-value of the rectangle origin, and the width and height.

Availability
Mac OS X v10.4 and later.

Declared In
`CIImageAccumulator.h`

format

Returns the pixel format of the image accumulator.

```
- (CIFormat)format
```

Return Value
The pixel format of the image accumulator.

Availability
Mac OS X v10.4 and later.

Declared In
`CIImageAccumulator.h`

image

Returns the current contents of the image accumulator.

```
- (CIImage *)image
```

Return Value
The image object that represents the current contents of the image accumulator.

Availability
Mac OS X v10.4 and later.

Declared In
`CIImageAccumulator.h`

initWithExtent:format:

Initializes an image accumulator with the specified extent and pixel format.

```
- (id)initWithExtent:(CGRect)r format:(CIFormat)f
```

Parameters

r
A rectangle that specifies the x-value of the rectangle origin, the y-value of the rectangle origin, and the width and height.

f
The format and size of each pixel. You must supply a pixel format constant, such as`kCIFormatARGB8` (32 bit-per-pixel, fixed-point pixel format) or `kCIFormatRGBAf` (128 bit-per-pixel, floating-point pixel format). See *CIImage Class Reference* for more information about pixel format constants.

Return Value
The initialized image accumulator object.

Availability
Mac OS X v10.4 and later.

See Also
`+ imageAccumulatorWithExtent:format:` (page 222)

Related Sample Code
CIAnnotation

Declared In
`CIImageAccumulator.h`

setImage:

Sets the contents of the image accumulator to the contents of the specified image object.

```
- (void)setImage:(CIImage *)im
```

Parameters

`im`
: The image object whose contents you want to assign to the image accumulator.

Availability
Mac OS X v10.4 and later.

See Also
`– setImage:dirtyRect:` (page 225)

Declared In
`CIImageAccumulator.h`

setImage:dirtyRect:

Updates an image accumulator with a subregion of an image object.

```
- (void)setImage:(CIImage *)im dirtyRect:(CGRect)r
```

Parameters

`im`
: The image object whose contents you want to assign to the image accumulator.

`r`
: A rectangle that defines the subregion of the image object that's changed since the last time you updated the image accumulator. You must guarantee that the new contents differ from the old only within the region specified by the this argument.

Discussion
For additional details on using this method, see "Imaging Dynamical Systems" in *Core Image Programming Guide*.

Availability
Mac OS X v10.4 and later.

See Also
`– setImage:` (page 224)

Declared In
`CIImageAccumulator.h`

CIKernel Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	Library/Frameworks/QuartzCore.framework
Declared in	QuartzCore/CIKernel.h
Availability	Mac OS X v10.4 and later
Companion guides	Core Image Programming Guide Core Image Kernel Language Reference
Related sample code	CIAnnotation

Overview

The `CIKernel` class maintains kernel routines that process individual pixels. The kernel routines in a `CIKernel` object use a subset of the OpenGL Shading Language and Core Image extensions to this language. You use a `CIKernel` object in conjunction with other Core Image classes, such as `CIFilter`, `CIFilterShape`, and `CISampler`, to create custom filters.

Tasks

Creating a Kernel

`+ kernelsWithString:` (page 228)
Creates and returns and array of `CIKernel` objects.

Getting a Kernel Name

`- name` (page 228)
Returns the name of a kernel routine.

Setting a Selector

- `setROISelector:` (page 229)
  Sets the selector used to query the region of interest of the kernel.

Class Methods

kernelsWithString:

Creates and returns and array of `CIKernel` objects.

```
+ (NSArray *)kernelsWithString:(NSString *)s
```

Parameters

s
: A program in the Core Image dialect of the OpenGL Shading Language that contains one or more routines, each of which is marked using the `kernel` keyword.

Return Value
An array of `CIKernel` objects. The array contains one `CIKernel` objects for each kernel routine in the supplied string.

Discussion
See *Core Image Kernel Language Reference* for more details.

Availability
Mac OS X v10.4 and later.

Related Sample Code
CIAnnotation

Declared In
`CIKernel.h`

Instance Methods

name

Returns the name of a kernel routine.

```
- (NSString *)name
```

Return Value
The name of the kernel routine.

Availability
Mac OS X v10.4 and later.

Declared In
`CIKernel.h`

setROISelector:

Sets the selector used to query the region of interest of the kernel.

```
- (void)setROISelector:(SEL)aMethod
```

Parameters

aMethod

A selector name.

Discussion
The `aMethod` argument must use the signature that is defined for the `regionOf:destRect:userInfo:` method, which is as follows:

```
- (CGRect) regionOf:(int)samplerIndex destRect:(CGRect)r userInfo:obj;
```

where:

- `samplerIndex` defines the sampler to query
- `destRect` is the extent of the region, in working space coordinates, to render.
- `userInfo` is the object associated with the `kCIApplyOptionUserInfo` option when the kernel is applied to its arguments. The `userInfo` is important because instance variables can't be used by the defining class. Instance variables must be passed through the `userInfo` argument.

The `regionOf:destRect:userInfo:` method of the CIFilter object is called by the framework. This method returns the rectangle that contains the region of the sampler that the kernel needs to render the specified destination rectangle.

A sample `regionOf:destRect:userInfo:` method might look as follows:

```
- (CGRect)regionOf:(int)sampler destRect:(CGRect)r userInfo:params
{
  float scale = fabs ([params X]);
  return CGRectInset (r, scale * -1.3333, scale * -1.3333);
}
```

In the filter code, you set the selector using the following:

```
kernel setROISelector:@selector(regionOf:destRect:userInfo:)]
```

Availability
Mac OS X v10.4 and later.

Related Sample Code
CIAnnotation

Declared In
`CIKernel.h`

CIPlugIn Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	Library/Frameworks/QuartzCore.framework
Declared in	QuartzCore/CIPlugIn.h
Availability	Mac OS X v10.4 and later
Companion guides	Image Unit Tutorial Core Image Programming Guide
Related sample code	CIAnnotation CIVideoDemoGL

Overview

The `CIPlugIn` class loads image units. An image unit is an image processing bundle that contains one or more Core Image filters. The `.plugin` extension indicates one or more filters that are packaged as an image unit.

Tasks

Loading Plug-ins

`+ loadAllPlugIns` (page 232)
Scans directories for files that have the `.plugin` extension and then loads the image units.

`+ loadNonExecutablePlugIns` (page 232)
Scans directories for files that have the `.plugin` extension and then loads only those filters that are marked by the image unit as non-executable filters.

`+ loadPlugIn:allowNonExecutable:` (page 232)
Loads filters from an image unit that have the appropriate executable status.

Class Methods

loadAllPlugIns

Scans directories for files that have the `.plugin` extension and then loads the image units.

```
+ (void)loadAllPlugIns
```

Discussion
This method scans the following directories:

- `/Library/Graphics/Image Units`
- `~/Library/Graphics/Image Units`

Call this method once. If you call this method more than once, Core Image loads newly added image units, but image units (and the filters they contain) that are already loaded are not removed.

Availability
Mac OS X v10.4 and later.

Related Sample Code
CIAnnotation
CIVideoDemoGL

Declared In
`CIPlugIn.h`

loadNonExecutablePlugIns

Scans directories for files that have the `.plugin` extension and then loads only those filters that are marked by the image unit as non-executable filters.

```
+ (void)loadNonExecutablePlugIns
```

Discussion
This call does not execute any of the code in the image unit, it simply loads the code. You need to call this method only once to load a specific image unit. The behavior of this method is not defined for multiple calls for the same image unit.

Availability
Mac OS X v10.4 and later.

Declared In
`CIPlugIn.h`

loadPlugIn:allowNonExecutable:

Loads filters from an image unit that have the appropriate executable status.

```
+ (void)loadPlugIn:(NSURL *)url allowNonExecutable:(BOOL)allowNonExecutable
```

Parameters

`url`

The location of the image unit to load.

`allowNonExecutable`

`TRUE` to load only those filters that are marked by the image unit as non-executable filters.

Discussion

You need to call this method only once to load a specific image unit. The behavior of this method is not defined for multiple calls for the same image unit.

Availability

Mac OS X v10.4 and later.

Related Sample Code

CIAnnotation

Declared In

`CIPlugIn.h`

CISampler Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSObject (NSObject)
Framework	Library/Frameworks/QuartzCore.framework
Declared in	QuartzCore/CISampler.h
Availability	Mac OS X v10.4 and later
Companion guide	Core Image Programming Guide
Related sample code	CIAnnotation

Overview

The `CISampler` class retrieves samples of images for processing by a `CIKernel` object. A `CISampler` object defines a coordinate transform, and modes for interpolation and wrapping. You use `CISampler` objects in conjunction with other Core Image classes, such as `CIFilter`, `CIKernel`, and `CIFilterShape`, to create custom filters.

Tasks

Creating a Sampler

`+ samplerWithImage:` (page 236)
    Creates and returns a sampler that references an image.

`+ samplerWithImage:keysAndValues:` (page 236)
    Creates and returns a sampler that references an image using options specified as key-value pairs.

`+ samplerWithImage:options:` (page 237)
    Creates and returns a sampler that references an image using options specified in a dictionary.

Initializing a Sampler

`- initWithImage:` (page 238)
: Initializes a sampler with an image object.

`- initWithImage:keysAndValues:` (page 239)
: Initializes the sampler with an image object using options specified as key-value pairs.

`- initWithImage:options:` (page 239)
: Initializes the sampler with an image object using options specified in a dictionary.

Getting Information About the Sampler Object

`- definition` (page 238)
: Gets the domain of definition (DOD) of the sampler.

`- extent` (page 238)
: Gets the rectangle that specifies the extent of the sampler.

Class Methods

samplerWithImage:

Creates and returns a sampler that references an image.

```
+ (CISampler *)samplerWithImage:(CIImage *)im
```

Parameters

im
: The image that you want the sampler to reference.

Return Value
A sampler object that references the image specified by the `im` argument.

Availability
Mac OS X v10.4 and later.

See Also
`+ samplerWithImage:keysAndValues:` (page 236)
`+ samplerWithImage:options:` (page 237)

Related Sample Code
CIAnnotation

Declared In
`CISampler.h`

samplerWithImage:keysAndValues:

Creates and returns a sampler that references an image using options specified as key-value pairs.

`+ (CISampler *)samplerWithImage:(CIImage *)im keysAndValues:key0, ...`

Parameters

`im`

The image that you want the sampler to reference.

`key0`

A list of key-value pairs that represent options. Each key needs to be followed by that appropriate value. You can supply one or more key-value pairs. Use `nil` to specify the end of the key-value options. See "`Sampler Option Keys`" (page 240).

Return Value

A sampler that references the image specified by the `im` argument and uses the specified options.

Availability

Mac OS X v10.4 and later.

See Also

`+ samplerWithImage:` (page 236)

`+ samplerWithImage:options:` (page 237)

Declared In

`CISampler.h`

samplerWithImage:options:

Creates and returns a sampler that references an image using options specified in a dictionary.

`+ (CISampler *)samplerWithImage:(CIImage *)im options:(NSDictionary *)dict`

Parameters

`im`

The image that you want the sampler to reference.

`dict`

A dictionary that contains options specified as key-value pairs. See "`Sampler Option Keys`" (page 240).

Return Value

A sampler that references the image specified by the `im` argument and uses the options specified in the dictionary.

Availability

Mac OS X v10.4 and later.

See Also

`+ samplerWithImage:` (page 236)

`+ samplerWithImage:keysAndValues:` (page 236)

Declared In

`CISampler.h`

Instance Methods

definition

Gets the domain of definition (DOD) of the sampler.

```
- (CIFilterShape *)definition
```

Return Value
The filter shape object that contains the DOD.

Discussion
The DOD contains all nontransparent pixels produced by referencing the sampler.

Availability
Mac OS X v10.4 and later.

Related Sample Code
CIAnnotation

Declared In
`CISampler.h`

extent

Gets the rectangle that specifies the extent of the sampler.

```
- (CGRect)extent
```

Return Value
The rectangle that specifies the area outside which the wrap mode set for the sampler is invoked.

Availability
Mac OS X v10.4 and later.

Related Sample Code
CIAnnotation

Declared In
`CISampler.h`

initWithImage:

Initializes a sampler with an image object.

```
- (id)initWithImage:(CIImage *)im
```

Parameters

`im`
The image object to initialize the sampler with.

Availability
Mac OS X v10.4 and later.

See Also
`- initWithImage:keysAndValues:` (page 239)
`- initWithImage:options:` (page 239)

Declared In
`CISampler.h`

initWithImage:keysAndValues:

Initializes the sampler with an image object using options specified as key-value pairs.

```
- (id)initWithImage:(CIImage *)im keysAndValues:key0, ...
```

Parameters

im
The image object to initialize the sampler with.

key0
A list of key-value pairs that represent options. Each key needs to be followed by that appropriate value. You can supply one or more key-value pairs. Use `nil` to specify the end of the key-value options. See "`Sampler Option Keys`" (page 240).

Availability
Mac OS X v10.4 and later.

See Also
`- initWithImage:` (page 238)
`- initWithImage:options:` (page 239)

Declared In
`CISampler.h`

initWithImage:options:

Initializes the sampler with an image object using options specified in a dictionary.

```
- (id)initWithImage:(CIImage *)im options:(NSDictionary *)dict
```

Parameters

im
The image to initialize the sampler with.

dict
A dictionary that contains options specified as key-value pairs. See "`Sampler Option Keys`" (page 240).

Availability
Mac OS X v10.4 and later.

See Also
`- initWithImage:` (page 238)

`- initWithImage:keysAndValues:` (page 239)

Declared In
`CISampler.h`

Constants

Sampler Option Keys

Keys for creating a sampler.

```
extern NSString *kCISamplerAffineMatrix;
extern NSString *kCISamplerWrapMode;
extern NSString *kCISamplerFilterMode
```

Constants

`kCISamplerAffineMatrix`

The key for an affine matrix. The associated value is an `NSArray` object ([*a b c d tx ty*]) that defines the transformation to apply to the sampler.

Available in Mac OS X v10.4 and later.

Declared in `CISampler.h`.

`kCISamplerWrapMode`

The key for the sampler wrap mode. The wrap mode specifies how Core Image produces pixels that are outside the extent of the sample. Possible values are `kCISamplerWrapBlack` (page 241) and `kCISamplerWrapClamp` (page 241).

Available in Mac OS X v10.4 and later.

Declared in `CISampler.h`.

`kCISamplerFilterMode`

The key for the filtering to use when sampling the image. Possible values are `kCISamplerFilterNearest` (page 241) and `kCISamplerFilterLinear` (page 241).

Available in Mac OS X v10.4 and later.

Declared in `CISampler.h`.

Declared In
`CISampler.h`

Sampler Option Values

Values for sampler option keys.

```
extern NSString *kCISamplerWrapBlack;
extern NSString *kCISamplerWrapClamp;
extern NSString *kCISamplerFilterNearest;
extern NSString *kCISamplerFilterLinear;
```

Constants

`kCISamplerWrapBlack`

Pixels are transparent black.

Available in Mac OS X v10.4 and later.

Declared in `CISampler.h`.

`kCISamplerWrapClamp`

Coordinates are clamped to the extent.

Available in Mac OS X v10.4 and later.

Declared in `CISampler.h`.

`kCISamplerFilterNearest`

Nearest neighbor sampling.

Available in Mac OS X v10.4 and later.

Declared in `CISampler.h`.

`kCISamplerFilterLinear`

Bilinear interpolation.

Available in Mac OS X v10.4 and later.

Declared in `CISampler.h`.

Declared In

`CISampler.h`

CIVector Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	Library/Frameworks/QuartzCore.framework
Declared in	QuartzCore/CIVector.h
Availability	Mac OS X v10.4 and later
Companion guide	Core Image Programming Guide
Related sample code	CarbonCocoaCoreImageTab CIAnnotation CITransitionSelectorSample2 CIVideoDemoGL Reducer

Overview

The `CIVector` class is used for coordinate values and direction vectors. You typically use a `CIVector` object to pass parameter values to Core Image filters. `CIVector` objects work in conjunction with other Core Image classes, such as `CIFilter`, `CIContext`, `CIImage`, and `CIColor`, to process images using the Core Image framework.

Tasks

Creating a Vector

`+ vectorWithValues:count:` (page 245)
    Creates and returns a vector that is initialized with the specified values.

`+ vectorWithX:` (page 246)
    Creates and returns a vector that is initialized with one value.

`+ vectorWithX:Y:` (page 246)
    Creates and returns a vector that is initialized with two values.

+ `vectorWithX:Y:Z:` (page 246)
  Creates and returns a vector that is initialized with three values.
+ `vectorWithX:Y:Z:W:` (page 247)
  Creates and returns a vector that is initialized with four values.
+ `vectorWithString:` (page 245)
  Creates and returns a vector that is initialized with values provided in a string representation.

Initializing a Vector

- `initWithValues:count:` (page 248)
  Initializes a vector with the provided values.
- `initWithX:` (page 249)
  Initializes the first position of a vector with the provided values.
- `initWithX:Y:` (page 249)
  Initializes the first two positions of a vector with the provided values.
- `initWithX:Y:Z:` (page 249)
  Initializes the first three positions of a vector with the provided values.
- `initWithX:Y:Z:W:` (page 250)
  Initializes four positions of a vector with the provided values.
- `initWithString:` (page 248)
  Initializes a vector with values provided in a string representation.

Getting Values From a Vector

- `valueAtIndex:` (page 251)
  Returns a value from a specific position in a vector.
- `count` (page 248)
  Returns the number of items in a vector.
- `X` (page 251)
  Returns the value located in the first position in a vector.
- `Y` (page 252)
  Returns the value located in the second position in a vector.
- `Z` (page 252)
  Returns the value located in the third position in a vector.
- `W` (page 251)
  Returns the value located in the fourth position in a vector.
- `stringRepresentation` (page 250)
  Returns a string representation for a vector.

Class Methods

vectorWithString:

Creates and returns a vector that is initialized with values provided in a string representation.

```
+ (CIVector *)vectorWithString:(NSString *)representation
```

Parameters

representation

A string that is in one of the formats returned by the `stringRepresentation` method.

Discussion

Some typical string representations for vectors are:

```
@"[1.0 0.5 0.3]"
```

which specifies a `vec3` vector whose components are `X = 1.0`, `Y = 0.5`, and `Z = 0.3`

```
@"[10.0 23.0]
```

which specifies a `vec2` vector show components are `X = 10.0` and `Y = 23.0`

Availability

Mac OS X v10.4 and later.

See Also

- `stringRepresentation` (page 250)

Declared In

`CIVector.h`

vectorWithValues:count:

Creates and returns a vector that is initialized with the specified values.

```
+ (CIVector *)vectorWithValues:(const CGFloat *)values count:(size_t)count
```

Parameters

values

The values to initialize the vector with.

count

The number of values in the vector.

Return Value

A vector initialized with the provided values.

Availability

Mac OS X v10.4 and later.

Declared In

`CIVector.h`

vectorWithX:

Creates and returns a vector that is initialized with one value.

```
+ (CIVector *)vectorWithX:(CGFloat)x
```

Parameters

x

The value to initialize the vector with.

Return Value
A vector initialized with the specified value.

Availability
Mac OS X v10.4 and later.

Declared In
`CIVector.h`

vectorWithX:Y:

Creates and returns a vector that is initialized with two values.

```
+ (CIVector *)vectorWithX:(CGFloat)x Y:(CGFloat)y
```

Parameters

x

The value for the first position in the vector.

y

The value for the second position in the vector.

Return Value
A vector initialized with the specified values.

Availability
Mac OS X v10.4 and later.

Related Sample Code
CarbonCocoaCoreImageTab
CIAnnotation
CIVideoDemoGL
Core Animation QuickTime Layer
Reducer

Declared In
`CIVector.h`

vectorWithX:Y:Z:

Creates and returns a vector that is initialized with three values.

```
+ (CIVector *)vectorWithX:(CGFloat)x Y:(CGFloat)y Z:(CGFloat)z
```

Parameters

x
: The value for the first position in the vector.

y
: The value for the second position in the vector.

z
: The value for the third position in the vector.

Return Value
A vector initialized with the specified values.

Availability
Mac OS X v10.4 and later.

Declared In
`CIVector.h`

vectorWithX:Y:Z:W:

Creates and returns a vector that is initialized with four values.

```
+ (CIVector *)vectorWithX:(CGFloat)x Y:(CGFloat)y Z:(CGFloat)z W:(CGFloat)w
```

Parameters

x
: The value for the first position in the vector.

y
: The value for the second position in the vector.

z
: The value for the third position in the vector.

w
: The value for the fourth position in the vector.

Return Value
A vector initialized with the specified values.

Availability
Mac OS X v10.4 and later.

Related Sample Code
CarbonCocoaCoreImageTab
CIAnnotation
CITransitionSelectorSample2
Reducer

Declared In
`CIVector.h`

Instance Methods

count

Returns the number of items in a vector.

```
- (size_t)count
```

Return Value
The number of items in the vector.

Availability
Mac OS X v10.4 and later.

Declared In
`CIVector.h`

initWithString:

Initializes a vector with values provided in a string representation.

```
- (id)initWithString:(NSString *)representation;
```

Parameters

`representation`
A string that is in one of the formats returned by the `stringRepresentation` method.

Availability
Mac OS X v10.4 and later.

See Also
- `stringRepresentation` (page 250)

Declared In
`CIVector.h`

initWithValues:count:

Initializes a vector with the provided values.

```
- (id)initWithValues:(const CGFloat *)values count:(size_t)count
```

Parameters

`values`
The values to initialize the vector with.

`count`
The number of values specified by the `values` argument.

Availability
Mac OS X v10.4 and later.

Declared In
`CIVector.h`

initWithX:

Initializes the first position of a vector with the provided values.

```
- (id)initWithX:(CGFloat)x
```

Parameters

x
The initialization value.

Availability
Mac OS X v10.4 and later.

Declared In
`CIVector.h`

initWithX:Y:

Initializes the first two positions of a vector with the provided values.

```
- (id)initWithX:(CGFloat)x Y:(CGFloat)y
```

Parameters

x
The initialization value for the first position.

y
The initialization value for the second position.

Availability
Mac OS X v10.4 and later.

Declared In
`CIVector.h`

initWithX:Y:Z:

Initializes the first three positions of a vector with the provided values.

```
- (id)initWithX:(CGFloat)x Y:(CGFloat)y Z:(CGFloat)z
```

Parameters

x
The initialization value for the first position.

y
The initialization value for the second position.

z
The initialization value for the third position.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CIVector.h`

initWithX:Y:Z:W:

Initializes four positions of a vector with the provided values.

```
- (id)initWithX:(CGFloat)x Y:(CGFloat)y Z:(CGFloat)z W:(CGFloat)w
```

Parameters

x
The initialization value for the first position.

y
The initialization value for the second position.

z
The initialization value for the third position.

w
The initialization value for the fourth position.

Availability
Mac OS X v10.4 and later.

Declared In
`CIVector.h`

stringRepresentation

Returns a string representation for a vector.

```
- (NSString *)stringRepresentation
```

Return Value
A string object.

Discussion
You convert the string representation returned by this method to a vector by supplying it as a parameter to the `vectorWithString:` method.

Some typical string representations for vectors are:

```
@"[1.0 0.5 0.3]"
```

which specifies a `vec3` vector whose components are `X = 1.0`, `Y = 0.5`, and `Z = 0.3`

```
@"[10.0 23.0]"
```

which specifies a `vec2` vector show components are `X = 10.0` and `Y = 23.0`

Availability
Mac OS X v10.4 and later.

See Also

`+ vectorWithString:` (page 245)

Declared In

`CIVector.h`

valueAtIndex:

Returns a value from a specific position in a vector.

```
- (CGFloat)valueAtIndex:(size_t)index
```

Parameters

`index`

The position in the vector of the value that you want to retrieve.

Return Value

The value retrieved from the vector or `0` if the position is undefined.

Discussion

The numbering of elements in a vector begins with zero.

Availability

Mac OS X v10.4 and later.

Declared In

`CIVector.h`

W

Returns the value located in the fourth position in a vector.

```
- (CGFloat)W
```

Return Value

The value retrieved from the vector.

Availability

Mac OS X v10.4 and later.

Declared In

`CIVector.h`

X

Returns the value located in the first position in a vector.

```
- (CGFloat)X
```

Return Value

The value retrieved from the vector.

Availability
Mac OS X v10.4 and later.

Related Sample Code
CIAnnotation

Declared In
`CIVector.h`

Y

Returns the value located in the second position in a vector.

```
- (CGFloat)Y
```

Return Value
The value retrieved from the vector.

Availability
Mac OS X v10.4 and later.

Related Sample Code
CIAnnotation

Declared In
`CIVector.h`

Z

Returns the value located in the third position in a vector.

```
- (CGFloat)Z
```

Return Value
The value retrieved from the vector.

Availability
Mac OS X v10.4 and later.

Declared In
`CIVector.h`

NSValue Core Animation Additions

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/QuartzCore.framework
Declared in	QuartzCore/CATransform3D.h
Companion guides	Core Animation Programming Guide Core Animation Cookbook

Overview

Core Animation adds two methods to the Foundation framework's `NSValue` class to support `CATransform3D` structure values.

Tasks

Creating an NSValue

`+ valueWithCATransform3D:` (page 253)
Creates and returns an NSValue object that contains a given `CATransform3D` structure.

Accessing Data

`- CATransform3DValue` (page 254)
Returns an `CATransform3D` structure representation of the receiver.

Class Methods

valueWithCATransform3D:

Creates and returns an NSValue object that contains a given `CATransform3D` structure.

```
+ (NSValue *)valueWithCATransform3D:(CATransform3D)aTransform
```

Parameters

aTransform

The value for the new object.

Return Value

A new `NSValue` object that contains the value of *aTransform*.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CATransform3D.h`

Instance Methods

CATransform3DValue

Returns an `CATransform3D` structure representation of the receiver.

```
- (CATransform3D)CATransform3DValue
```

Return Value

An `CATransform3D` structure representation of the receiver.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`CATransform3D.h`

Protocols

CAAction Protocol Reference

Adopted by	CAAnimation
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	CALayer.h
Companion guides	Core Animation Programming Guide Core Animation Cookbook

Overview

The `CAAction` protocol provides an interface that allows an object to respond to an action triggered by an `CALayer`. When queried with an action identifier (a key path, an external action name, or a predefined action identifier) the layer returns the appropriate action object–which must implement the `CAAction` protocol–and sends it a `runActionForKey:object:arguments:` (page 257) message.

Tasks

Responding to an Action

`– runActionForKey:object:arguments:` (page 257)
Called to trigger the action specified by the identifier.

Instance Methods

runActionForKey:object:arguments:

Called to trigger the action specified by the identifier.

```
- (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

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

Overview

`CALayoutManager` is an informal protocol implemented by Core Animation layout managers. If a layer's sublayers require custom layout you create a class that implements this protocol and set it as the layer's layout manager using the `CALayer` method `setLayoutManager:`. Your custom layout manager is then used when the layer invokes `setNeedsLayout` (page 79) or `layoutSublayers` (page 73).

Tasks

Layout Layers

`– invalidateLayoutOfLayer:` (page 259)
Invalidates the layout of the specified layer.

`– layoutSublayersOfLayer:` (page 260)
Layout each of the sublayers in the specified layer.

Calculate Layer Size

`– preferredSizeOfLayer:` (page 260)
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

Adopted by	CAAnimation CALayer
Framework	/System/Library/Frameworks/QuartzCore.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	CAMediaTiming.h
Companion guides	Core Animation Programming Guide Core Animation Cookbook

Overview

The `CAMediaTiming` protocol models a hierarchical timing system, with each object describing the mapping of time values from the object's parent to local time.

Absolute time is defined as mach time converted to seconds. The `CACurrentMediaTime` (page 362) 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 264) *property*
Specifies the begin time of the receiver in relation to its parent object, if applicable.

`timeOffset` (page 266) *property*
Specifies an additional time offset in active local time.

Repeating Animations

`repeatCount` (page 265) *property*
Determines the number of times the animation will repeat.

`repeatDuration` (page 266) *property*
Determines how many seconds the animation will repeat for.

Duration and Speed

`duration` (page 265) *property*
Specifies the basic duration of the animation, in seconds.

`speed` (page 266) *property*
Specifies how time is mapped to receiver's time space from the parent time space.

Playback Modes

`autoreverses` (page 264) *property*
Determines if the receiver plays in the reverse upon completion.

`fillMode` (page 265) *property*
Determines if the receiver's presentation is frozen or removed once its active duration has completed.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

autoreverses

Determines if the receiver plays in the reverse upon completion.

```
@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.

Declared In
`CAMediaTiming.h`

beginTime

Specifies the begin time of the receiver in relation to its parent object, if applicable.

```
@property CFTimeInterval beginTime
```

Discussion
Defaults to 0.

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

Declared In
`CAMediaTiming.h`

duration

Specifies the basic duration of the animation, in seconds.

```
@property CFTimeInterval duration
```

Discussion
Defaults to 0.

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

Declared In
`CAMediaTiming.h`

fillMode

Determines if the receiver's presentation is frozen or removed once its active duration has completed.

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

Discussion
The possible values are described in "`Fill Modes`" (page 267). The default is `kCAFillModeRemoved` (page 267).

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

Declared In
`CAMediaTiming.h`

repeatCount

Determines the number of times the animation will repeat.

```
@property float repeatCount
```

Discussion
May be fractional. If the `repeatCount` is 0, it is ignored. Defaults to 0. If both `repeatDuration` (page 266) and `repeatCount` (page 265) are specified the behavior is undefined.

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

Declared In
`CAMediaTiming.h`

repeatDuration

Determines how many seconds the animation will repeat for.

```
@property CFTimeInterval repeatDuration
```

Discussion
Defaults to 0. If the `repeatDuration` is 0, it is ignored. If both `repeatDuration` (page 266) and `repeatCount` (page 265) 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.

```
@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.

Declared In
`CAMediaTiming.h`

timeOffset

Specifies an additional time offset in active local time.

```
@property CFTimeInterval 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 265) 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`

CIImageProvider Protocol Reference

(informal protocol)

Adopted by	NSObject
Framework	Library/Frameworks/QuartzCore.framework
Declared in	QuartzCore/CIImageProvider.h
Availability	Available in Mac OS X v10.4 and later
Companion guide	Core Image Programming Guide

Overview

The `CIImageProvider` informal protocol defines methods for supplying bitmap data to create or initialize a `CIImage` object.

Tasks

Providing Image Data

`- provideImageData:bytesPerRow:origin:size:userInfo:` (page 269)
    Supplies data to a `CIImage` object.

Instance Methods

provideImageData:bytesPerRow:origin:size:userInfo:

Supplies data to a `CIImage` object.

```
- (void)provideImageData:(void *)data bytesPerRow:(size_t)rowbytes  origin:(size_t)
    x:(size_t)y size:(size_t)width:(size_t)height userInfo:(id)info
```

Parameters

data
    A pointer to image data. Note that `data[0]` refers to the first byte of the requested subimage, not the larger image buffer.

rowbytes
: The number of bytes per row.

x
: The x origin of the image data.

y
: The y origin of the image data.

width
: The width of the image data.

height
: The height of the image data.

info
: User supplied data, which is optional.

Discussion

You can supply the image provider to these methods of the `CIImage` class:

- `imageWithImageProvider:size::format:colorSpace:options:` to create a CIImage object from image data
- `initWithImageProvider:size::format:colorSpace:options:` to initialize an existing CIImage with data

You initialize the given bitmap with the subregion specified by the arguments `x`, `y`, `width`, and `height`. The subregion uses the local coordinate space of the image, with the origin at the upper-left corner of the image. If you change the virtual memory mapping of the buffer specified by the `data` argument (such as by using `vm_copy` to modify it), the behavior is undefined.

That this callback always requests the full image data regardless of what is actually visible. All of the image is loaded or none of it is. The exception is when you create a tiled image by specifying the `kCIImageProviderTileSize` option. In this case, only the needed tiles are requested.

Availability

Mac OS X v10.4 and later.

Constants

Image Provider Options

Keys for the options dictionary of an image provider.

```
extern NSString *kCIImageProviderTileSize;
extern NSString *kCIImageProviderUserInfo;
```

Constants

`kCIImageProviderTileSize`
: A key for the image tiles size. The associated value is an `NSArray` that contains`NSNumber` objects for the dimensions of the image tiles requested from the image provider.

  Available in Mac OS X v10.4 and later.

  Declared in `CIImageProvider.h`.

`kCIImageProviderUserInfo`
: A key for data needed by the image provider. The associated value is an object that contains the needed data.

  Available in Mac OS X v10.4 and later.

  Declared in `CIImageProvider.h`.

Discussion
You can use these options when you create or initialize an image provider with such methods as `imageWithImageProvider:size::format:colorSpace:options:` or `initWithImageProvider:size::format:colorSpace:options:`.

Declared In
`CIImageProvider.h`

CIPlugInRegistration Protocol Reference

Adopted by	CIPlugIn
Framework	Library/Frameworks/QuartzCore.framework
Declared in	QuartzCore/CIPlugInInterface.h
Availability	Mac OS X v10.4 and later
Companion guides	Image Unit Tutorial Core Image Programming Guide

Overview

The `CIPlugInRegistration` protocol defines a method for loading Core Image image units. The principal class of an image unit bundle must support this protocol.

Tasks

Initializing Plug-ins

`– load:` (page 273)
Loads and initializes an image unit, performing custom tasks as needed.

Instance Methods

load:

Loads and initializes an image unit, performing custom tasks as needed.

```
- (BOOL)load:(void *)host
```

Parameters

`host`
Reserved for future use.

Return Value
Returns `true` if the image unit is successfully initialized

Discussion
The `load` method is called once by the host to initialize the image unit when the first filter in the image unit is instantiated. The method provides the image unit with an opportunity to perform custom initialization, such as a registration check.

Availability
Mac OS X v10.4 and later.

Declared In
`CIPlugInInterface.h`

Other References

Core Video Reference

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

Overview

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

Core Video is available in:

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

Functions by Task

CVBuffer Functions

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

`CVBufferGetAttachment` (page 283)
    Returns a specific attachment of a Core Video buffer.

`CVBufferGetAttachments` (page 284)
    Returns all attachments of a Core Video buffer.

`CVBufferPropagateAttachments` (page 284)
: Copies all propagatable attachments from one Core Video buffer to another.

`CVBufferRelease` (page 285)
: Releases a Core Video buffer.

`CVBufferRemoveAllAttachments` (page 285)
: Removes all attachments of a Core Video buffer.

`CVBufferRemoveAttachment` (page 286)
: Removes a specific attachment of a Core Video buffer.

`CVBufferRetain` (page 286)
: Retains a Core Video buffer.

`CVBufferSetAttachment` (page 287)
: Sets or adds an attachment of a Core Video buffer.

`CVBufferSetAttachments` (page 288)
: Sets a set of attachments for a Core Video buffer.

CVDisplayLink Functions

The main purpose of the CoreVideo display link to provide a separate high-priority thread to notify your application when a given display will need each frame. How often a frame is requested is based on the refresh rate of the display device currently associated with the display link. A CoreVideo display link is represented in code by the `CVDisplayLinkRef` type. The display link API uses the Core Foundation class system internally to provide reference counting behaviour and other useful properties.

`CVDisplayLinkCreateWithCGDisplay` (page 289)
: Creates a display link for a single display.

`CVDisplayLinkCreateWithActiveCGDisplays` (page 288)
: Creates a display link capable of being used with all active displays.

`CVDisplayLinkCreateWithCGDisplays` (page 289)
: Creates a display link for an array of displays.

`CVDisplayLinkCreateWithOpenGLDisplayMask` (page 290)
: Creates a display link from an OpenGL display mask.

`CVDisplayLinkGetActualOutputVideoRefreshPeriod` (page 290)
: Retrieves the actual output refresh period of a display as measured by the host time base.

`CVDisplayLinkGetCurrentCGDisplay` (page 291)
: Gets the current display associated with a display link.

`CVDisplayLinkGetCurrentTime` (page 291)
: Retrieves the current (“now”) time of a given display link.

`CVDisplayLinkGetNominalOutputVideoRefreshPeriod` (page 292)
: Retrieves the nominal refresh period of a display link.

`CVDisplayLinkGetOutputVideoLatency` (page 292)
: Retrieves the nominal latency of a display link.

`CVDisplayLinkGetTypeID` (page 293)
: Obtains the Core Foundation ID for the display link data type.

`CVDisplayLinkIsRunning` (page 293)
: Indicates whether a given display link is running.

`CVDisplayLinkRelease` (page 294)
    Releases a display link.

`CVDisplayLinkRetain` (page 294)
    Retains a display link.

`CVDisplayLinkSetCurrentCGDisplay` (page 294)
    Sets the current display of a display link.

`CVDisplayLinkSetCurrentCGDisplayFromOpenGLContext` (page 295)
    Selects the display link most optimal for the current renderer of an OpenGL context.

`CVDisplayLinkSetOutputCallback` (page 296)
    Set the renderer output callback function.

`CVDisplayLinkStart` (page 297)
    Activates a display link.

`CVDisplayLinkStop` (page 297)
    Stops a display link.

`CVDisplayLinkTranslateTime` (page 298)
    Translates the time in the display link's time base from one representation to another.

CVHostTime Functions

`CVGetCurrentHostTime` (page 298)
    Retrieves the current value of the host time base.

`CVGetHostClockFrequency` (page 299)
    Retrieve the frequency of the host time base.

`CVGetHostClockMinimumTimeDelta` (page 299)
    Retrieve the smallest possible increment in the host time base.

CVImageBuffer Functions

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

`CVImageBufferGetCleanRect` (page 299)
    Returns the source rectangle of a Core Video image buffer that represents the clean aperture of the buffer in encoded pixels.

`CVImageBufferGetColorSpace` (page 300)
    Returns the color space of a Core Video image buffer.

`CVImageBufferGetDisplaySize` (page 300)
    Returns the nominal output display size, in square pixels, of a Core Video image buffer.

`CVImageBufferGetEncodedSize` (page 301)
    Returns the full encoded dimensions of a Core Video image buffer.

CVOpenGLBuffer Functions

The Core Video OpenGL buffer (type `CVOpenGLBufferRef` is a wrapper around the standard OpenGL pbuffer.

`CVOpenGLBufferAttach` (page 301)
: Attaches an OpenGL context to a Core Video OpenGL buffer.

`CVOpenGLBufferCreate` (page 302)
: Create a new Core Video OpenGL buffer that can be used for OpenGL rendering purposes

`CVOpenGLBufferGetAttributes` (page 303)
: Obtains the attributes of a Core Video OpenGL buffer.

`CVOpenGLBufferGetTypeID` (page 303)
: Obtains the Core Foundation type ID for the OpenGL buffer type.

`CVOpenGLBufferRelease` (page 307)
: Releases a Core Video OpenGL buffer.

`CVOpenGLBufferRetain` (page 307)
: Retains a Core Video OpenGL buffer.

CVOpenGLBufferPool Functions

An OpenGL buffer pool is a utility object for managing a set of OpenGL buffer objects for recycling.

`CVOpenGLBufferPoolCreate` (page 304)
: Creates a new OpenGL buffer pool.

`CVOpenGLBufferPoolCreateOpenGLBuffer` (page 304)
: Creates a new OpenGL buffer from an OpenGL buffer pool.

`CVOpenGLBufferPoolGetAttributes` (page 305)
: Returns the pool attributes dictionary for an Open GL buffer pool.

`CVOpenGLBufferPoolGetOpenGLBufferAttributes` (page 305)
: Returns the attributes of OpenGL buffers that will be created from a buffer pool.

`CVOpenGLBufferPoolGetTypeID` (page 306)
: Obtains the Core Foundation ID for the OpenGL buffer pool type.

`CVOpenGLBufferPoolRelease` (page 306)
: Releases an OpenGL buffer pool.

`CVOpenGLBufferPoolRetain` (page 306)
: Retains an OpenGL buffer pool.

CVOpenGLTexture Functions

The Core Video OpenGL texture is a wrapper around the standard OpenGL texture type.

`CVOpenGLTextureGetCleanTexCoords` (page 311)
: Returns the texture coordinates for the part of the image that should be displayed.

`CVOpenGLTextureGetName` (page 312)
: Returns the texture target name of a CoreVideo OpenGL texture.

`CVOpenGLTextureGetTarget` (page 312)
: Returns the texture target (for example, `GL_TEXTURE_2D`) of an OpenGL texture.

`CVOpenGLTextureGetTypeID` (page 313)
: Obtains the Core Foundation ID for the Core Video OpenGL texture type.

`CVOpenGLTextureIsFlipped` (page 313)
: Determines whether or not an OpenGL texture is flipped vertically.

`CVOpenGLTextureRelease` (page 314)
: Releases a Core Video OpenGL texture.

`CVOpenGLTextureRetain` (page 314)
: Retains a Core Video OpenGL texture.

CVOpenGLTextureCache Functions

`CVOpenGLTextureCacheCreate` (page 308)
: Creates an OpenGL texture cache.

`CVOpenGLTextureCacheCreateTextureFromImage` (page 308)
: Creates an OpenGL texture object from an existing image buffer.

`CVOpenGLTextureCacheFlush` (page 309)
: Flushes the OpenGL texture cache.

`CVOpenGLTextureCacheGetTypeID` (page 310)
: Returns the Core Foundation ID of the texture cache type.

`CVOpenGLTextureCacheRelease` (page 310)
: Releases an OpenGL texture cache.

`CVOpenGLTextureCacheRetain` (page 310)
: Retains an OpenGL texture cache.

CVPixelBuffer Functions

A pixel buffer stores an image in main memory

`CVPixelBufferCreate` (page 315)
: Creates a single pixel buffer for a given size and pixel format.

`CVPixelBufferCreateResolvedAttributesDictionary` (page 316)
: Takes an array of `CFDictionary` objects describing various pixel buffer attributes and tries to resolve them into a single dictionary.

`CVPixelBufferCreateWithBytes` (page 316)
: Creates a pixel buffer for a given size and pixel format containing data specified by a memory location.

`CVPixelBufferCreateWithPlanarBytes` (page 317)
: Creates a single pixel buffer in planar format for a given size and pixel format containing data specified by a memory location.

`CVPixelBufferFillExtendedPixels` (page 319)
: Fills the extended pixels of the pixel buffer.

`CVPixelBufferGetBaseAddress` (page 319)
: Returns the base address of the pixel buffer.

`CVPixelBufferGetBaseAddressOfPlane` (page 320)
: Returns the base address of the plane at the specified plane index.

`CVPixelBufferGetBytesPerRow` (page 320)
: Returns the number of bytes per row of the pixel buffer.

`CVPixelBufferGetBytesPerRowOfPlane` (page 321)
: Returns the number of bytes per row for a plane at the specified index in the pixel buffer.

`CVPixelBufferGetDataSize` (page 321)
: Returns the data size for contiguous planes of the pixel buffer.

`CVPixelBufferGetExtendedPixels` (page 322)
: Returns the amount of extended pixel padding in the pixel buffer.

`CVPixelBufferGetHeight` (page 322)
: Returns the height of the pixel buffer.

`CVPixelBufferGetHeightOfPlane` (page 323)
: Returns the height of the plane at planeIndex in the pixel buffer.

`CVPixelBufferGetPixelFormatType` (page 323)
: Returns the pixel format type of the pixel buffer.

`CVPixelBufferGetPlaneCount` (page 324)
: Returns number of planes of the pixel buffer.

`CVPixelBufferGetTypeID` (page 324)
: Returns the Core Foundation ID of the pixel buffer type.

`CVPixelBufferGetWidth` (page 325)
: Returns the width of the pixel buffer.

`CVPixelBufferGetWidthOfPlane` (page 325)
: Returns the width of the plane at a given index in the pixel buffer.

`CVPixelBufferIsPlanar` (page 326)
: Determine if the pixel buffer is planar.

`CVPixelBufferLockBaseAddress` (page 326)
: Locks the base address of the pixel buffer.

`CVPixelBufferRelease` (page 330)
: Releases a pixel buffer.

`CVPixelBufferRetain` (page 331)
: Retains a pixel buffer.

`CVPixelBufferUnlockBaseAddress` (page 331)
: Unlocks the base address of the pixel buffer.

CVPixelBufferPool Functions

`CVPixelBufferPoolCreate` (page 327)
: Creates a pixel buffer pool.

`CVPixelBufferPoolCreatePixelBuffer` (page 327)
: Creates a pixel buffer from a pixel buffer pool.

`CVPixelBufferPoolGetAttributes` (page 328)
: Returns the pool attributes dictionary for a pixel buffer pool.

`CVPixelBufferPoolGetPixelBufferAttributes` (page 328)
: Returns the attributes of pixel buffers that will be created from this pool.

`CVPixelBufferPoolGetTypeID` (page 329)
: Returns the Core Foundation ID of the pixel buffer pool type.

`CVPixelBufferPoolRelease` (page 329)
: Releases a pixel buffer pool.

`CVPixelBufferPoolRetain` (page 330)
: Retains a pixel buffer pool.

CVPixelFormatDescription Functions

Used only if you are defining a custom pixel format.

`CVPixelFormatDescriptionRegisterDescriptionWithPixelFormatType` (page 333)
: Registers a pixel format description with Core Video.

`CVPixelFormatDescriptionCreateWithPixelFormatType` (page 332)
: Creates a pixel format description from a given `OSType` identifier.

`CVPixelFormatDescriptionArrayCreateWithAllPixelFormatTypes` (page 332)
: Returns all the pixel format descriptions known to Core Video.

Functions

CVBufferGetAttachment

Returns a specific attachment of a Core Video buffer.

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

Parameters

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

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

attachmentMode
: On return, `attachmentMode` points to the mode of the attachment. See "CVBuffer Attachment Modes" (page 343) for possible values. If the attachment mode is not defined, this parameter returns `NULL`.

Return Value
If found, the specified attachment.

Discussion
You can attach any Core Foundation object to a Core Video buffer to store additional information by calling `CVBufferSetAttachment` (page 287) or `CVBufferSetAttachments` (page 288).

You can find predefined attachment keys in "CVBuffer Attachment Keys" (page 343) and "Image Buffer Attachment Keys" (page 347).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVBuffer.h`

CVBufferGetAttachments

Returns all attachments of a Core Video buffer.

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

Parameters

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

attachmentMode
The mode of the attachments you want to obtain. See "CVBuffer Attachment Modes" (page 343) for possible values.

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

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

You can find predefined attachment keys in "CVBuffer Attachment Keys" (page 343) and "Image Buffer Attachment Keys" (page 347).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVBuffer.h`

CVBufferPropagateAttachments

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

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

Parameters

sourceBuffer
The buffer to copy attachments from.

destinationBuffer
The buffer to copy attachments to.

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVBuffer.h`

CVBufferRelease

Releases a Core Video buffer.

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

Parameters

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

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

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie
MovieVideoChart
VideoViewer

Declared In
`CVBuffer.h`

CVBufferRemoveAllAttachments

Removes all attachments of a Core Video buffer.

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

Parameters

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

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVBuffer.h`

CVBufferRemoveAttachment

Removes a specific attachment of a Core Video buffer.

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

Parameters

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

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

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

You can find predefined attachment keys in "CVBuffer Attachment Keys" (page 343) and "Image Buffer Attachment Keys" (page 347).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVBuffer.h`

CVBufferRetain

Retains a Core Video buffer.

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

Parameters

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

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

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

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie
MovieVideoChart

Declared In
`CVBuffer.h`

CVBufferSetAttachment

Sets or adds an attachment of a Core Video buffer.

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

Parameters

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

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

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

attachmentMode
Specifies the attachment mode for this attachment. See "CVBuffer Attachment Modes" (page 343) for possible values. Any given attachment key may exist in only one mode at a time.

Discussion
You can attach any Core Foundation object to a Core Video buffer to store additional information. If the key doesn't currently exist for the buffer object when you call this function, the new attachment will be added. If the key does exist, the existing attachment will be replaced. In both cases the retain count of the attachment will be incremented. The value can be any CFType. You can find predefined attachment keys in "CVBuffer Attachment Keys" (page 343) and "Image Buffer Attachment Keys" (page 347).

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

To retrieve attachments, use the `CVBufferGetAttachment` (page 283) or `CVBufferGetAttachments` (page 284) functions.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVBuffer.h`

CVBufferSetAttachments

Sets a set of attachments for a Core Video buffer.

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

Parameters

buffer

The Core Video buffer to which to set the attachments.

theAttachments

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

attachmentMode

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

Discussion

`CVBufferSetAttachments` is a convenience call that in turn calls `CVBufferSetAttachment` (page 287) for each key and value in the given dictionary. All key-value pairs must be in the root level of the dictionary.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVBuffer.h`

CVDisplayLinkCreateWithActiveCGDisplays

Creates a display link capable of being used with all active displays.

```
CVReturn CVDisplayLinkCreateWithActiveCGDisplays (
   CVDisplayLinkRef *displayLinkOut
);
```

Parameters

displayLinkOut

On return, `displayLinkOut` points to the newly created display link.

Return Value

A Core Video result code. See "Result Codes" (page 358) for possible values.

Discussion

`CVDisplayLinkCreateWithActiveCGDisplays` determines the displays actively used by the host computer and creates a display link compatible with all of them. For most applications, calling this function is the most convenient way to create a display link. After creation, you can assign the display link to any active display by calling `CVDisplayLinkSetCurrentCGDisplay` (page 294).

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

QTQuartzPlayer

VideoViewer

Declared In
`CVDisplayLink.h`

CVDisplayLinkCreateWithCGDisplay

Creates a display link for a single display.

```
CVReturn CVDisplayLinkCreateWithCGDisplay (
   CGDirectDisplayID displayID,
   CVDisplayLinkRef *displayLinkOut
);
```

Parameters

`displayID`
The Core Graphics ID of the target display.

`displayLinkOut`
On return, `displayLinkOut` points to the newly created display link.

Return Value
A Core Video result code. See "Result Codes" (page 358) for possible values.

Discussion
Use this call to create a display link for a single display.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
OpenGLCaptureToMovie
QTCoreImage101
QTCoreVideo102
QTCoreVideo201
QTCoreVideo301

Declared In
`CVDisplayLink.h`

CVDisplayLinkCreateWithCGDisplays

Creates a display link for an array of displays.

```
CVReturn CVDisplayLinkCreateWithCGDisplays (
   CGDirectDisplayID *displayArray,
   CFIndex count,
   CVDisplayLinkRef *displayLinkOut
);
```

Parameters

`displayArray`
A pointer to an array of Core Graphics display IDs representing all the active monitors you want to use with this display link.

count

The number of displays in the display array.

displayLisk

On return, `displayLinkOut` points to the newly created display link.

Return Value

A Core Video result code. See "Result Codes" (page 358) for possible values.

Discussion

Use this call to create a display link for a set of displays identified by the Core Graphics display IDs.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVDisplayLink.h`

CVDisplayLinkCreateWithOpenGLDisplayMask

Creates a display link from an OpenGL display mask.

```
CVReturn CVDisplayLinkCreateWithOpenGLDisplayMask (
   CGOpenGLDisplayMask mask,
   CVDisplayLinkRef *displayLinkOut
);
```

Parameters

mask

The OpenGL display mask describing the available displays.

displayLinkOut

On return, `displayLinkOut` points to the newly created display link.

Return Value

A Core Video result code. See "Result Codes" (page 358) for possible values.

Discussion

Using this function avoids having to call the Core Graphics function `CGOpenGLDisplayMaskToDisplayID`.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CIVideoDemoGL

LiveVideoMixer2

LiveVideoMixer3

Declared In

`CVDisplayLink.h`

CVDisplayLinkGetActualOutputVideoRefreshPeriod

Retrieves the actual output refresh period of a display as measured by the host time base.

```
double CVDisplayLinkGetActualOutputVideoRefreshPeriod (
   CVDisplayLinkRef displayLink
);
```

Parameters

displayLink
The display link to get the refresh period from.

Return Value
A double-precision floating-point value representing the actual refresh period. This value may be zero if the device is not running or is otherwise unavailable.

Discussion
This call returns the actual output refresh period (in seconds) as computed relative to the host time base.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVDisplayLink.h`

CVDisplayLinkGetCurrentCGDisplay

Gets the current display associated with a display link.

```
CGDirectDisplayID CVDisplayLinkGetCurrentCGDisplay (
   CVDisplayLinkRef displayLink
);
```

Parameters

displayLink
The display link whose current display you want obtain.

Return Value
A `CGDirectDisplayID` representing the current display.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVDisplayLink.h`

CVDisplayLinkGetCurrentTime

Retrieves the current (“now”) time of a given display link.

```
CVReturn CVDisplayLinkGetCurrentTime (
   CVDisplayLinkRef displayLink,
   CVTimeStamp *outTime
);
```

Parameters

displayLink
The display link whose current time you want to obtain.

outTime

A pointer to a `CVTimeStamp` structure. Note that yout must set the version in the structure (currently 0) before calling to indicate which version of the timestamp structure you want.

Return Value

A Core Video result code. See "Result Codes" (page 358) for possible values.

Discussion

You use this call to obtain the timestamp of the frame that is currently being displayed.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVDisplayLink.h`

CVDisplayLinkGetNominalOutputVideoRefreshPeriod

Retrieves the nominal refresh period of a display link.

```
CVTime CVDisplayLinkGetNominalOutputVideoRefreshPeriod (
   CVDisplayLinkRef displayLink
);
```

Parameters

displayLink

The display link whose refresh period you want to obtain.

Return Value

A `CVTime` structure that holds the nominal refresh period. This value is indefinite if an invalid display link was specified.

Discussion

This call allows one to retrieve the device's ideal refresh period. For example, an NTSC output device might report 1001/60000 to represent the exact NTSC vertical refresh rate.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVDisplayLink.h`

CVDisplayLinkGetOutputVideoLatency

Retrieves the nominal latency of a display link.

```
CVTime CVDisplayLinkGetOutputVideoLatency (
   CVDisplayLinkRef displayLink
);
```

Parameters

displayLink

The display link whose latency value you want to obtain.

Return Value
A `CVTime` structure that holds the latency value. This value may be indefinite.

Discussion
This call allows you to retrieve the device's built-in output latency. For example, an NTSC device with one frame of latency might report back 1001/30000 or 2002/60000.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVDisplayLink.h`

CVDisplayLinkGetTypeID

Obtains the Core Foundation ID for the display link data type.

```
CFTypeID CVDisplayLinkGetTypeID (
   void
);
```

Return Value
The Core Foundation ID for this type.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVDisplayLink.h`

CVDisplayLinkIsRunning

Indicates whether a given display link is running.

```
Boolean CVDisplayLinkIsRunning (
   CVDisplayLinkRef displayLink
);
```

Parameters

`displayLink`
    The display link whose run state you want to determine.

Return Value
Returns `true` if the display link is running, `false` otherwise.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CIVideoDemoGL

QTCoreImage101

QTCoreVideo102

QTCoreVideo201

QTCoreVideo301

Declared In
`CVDisplayLink.h`

CVDisplayLinkRelease

Releases a display link.

```
void CVDisplayLinkRelease (
   CVDisplayLinkRef displayLink
);
```

Parameters

`displayLink`
The display link to release. This function is *`NULL`*-safe.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
LiveVideoMixer3
QTCoreImage101
QTCoreVideo102
QTCoreVideo201
QTCoreVideo301

Declared In
`CVDisplayLink.h`

CVDisplayLinkRetain

Retains a display link.

```
CVDisplayLinkRef CVDisplayLinkRetain (
   CVDisplayLinkRef displayLink
);
```

Parameters

`displayLink`
The display link to retain. This function is *`NULL`*-safe.

Return Value
For convenience, this function returns the retained display link if successful.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVDisplayLink.h`

CVDisplayLinkSetCurrentCGDisplay

Sets the current display of a display link.

```
CVReturn CVDisplayLinkSetCurrentCGDisplay (
   CVDisplayLinkRef displayLink,
   CGDirectDisplayID displayID
);
```

Parameters

displayLink
: The display link whose display you want to set.

displayID
: The ID of the display to be set.

Return Value
A Core Video result code. See "Result Codes" (page 358) for possible values.

Discussion
Although it is safe to call this function on a running display link, a discontinuity may appear in the video timestamp.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
LiveVideoMixer3

QTCoreImage101

QTCoreVideo102

QTCoreVideo201

QTCoreVideo301

Declared In
`CVDisplayLink.h`

CVDisplayLinkSetCurrentCGDisplayFromOpenGLContext

Selects the display link most optimal for the current renderer of an OpenGL context.

```
CVReturn CVDisplayLinkSetCurrentCGDisplayFromOpenGLContext (
   CVDisplayLinkRef displayLink,
   CGLContextObj cglContext,
   CGLPixelFormatObj cglPixelFormat
);
```

Parameters

displayLink
: The display link for which you want to set the current display.

cglContext
: The OpenGL context to retrieve the current renderer from.

cglPixelFormat
: The OpenGL pixel format used to create the passed-in OpenGL context.

Return Value
A Core Video result code. See "Result Codes" (page 358) for possible values.

Discussion
This function chooses the display with the lowest refresh rate.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
VideoViewer

Declared In
`CVDisplayLink.h`

CVDisplayLinkSetOutputCallback

Set the renderer output callback function.

```
CVReturn CVDisplayLinkSetOutputCallback (
   CVDisplayLinkRef displayLink,
   CVDisplayLinkOutputCallback callback,
   void *userInfo
);
```

Parameters

displayLink
The display link whose output callback you want to set.

callback
The callback function to set for this display link. See `CVDisplayLinkOutputCallback` (page 333) for more information about implementing this function.

userInfo
A pointer to user data.

Return Value
A Core Video result code. See "Result Codes" (page 358) for possible values.

Discussion
The display link invokes this callback whenever it wants you to output a frame.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
LiveVideoMixer3

QTCoreImage101

QTCoreVideo102

QTCoreVideo201

QTCoreVideo301

Declared In
`CVDisplayLink.h`

CVDisplayLinkStart

Activates a display link.

```
CVReturn CVDisplayLinkStart (
   CVDisplayLinkRef displayLink
);
```

Parameters

displayLink

The display link to activate.

Return Value
A Core Video result code. See "Result Codes" (page 358) for possible values.

Discussion
Calling this function starts the display link thread, which then periodically calls back to your application to request that you display frames. If the specified display link is already running, `CVDisplayLinkStart` returns an error.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTCoreVideo102

QTCoreVideo103

QTCoreVideo201

QTCoreVideo202

QTCoreVideo301

Declared In
`CVDisplayLink.h`

CVDisplayLinkStop

Stops a display link.

```
CVReturn CVDisplayLinkStop (
   CVDisplayLinkRef displayLink
);
```

Parameters

displayLink

The display link to stop.

Return Value
A Core Video result code. See "Result Codes" (page 358) for possible values.

Discussion
If the specified display link is already stopped, `CVDisplayLinkStop` returns an error.

In Mac OS X v.10.4 and later, the display link thread is automatically stopped if the user employs Fast User Switching. The display link is restarted when switching back to the original user.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code

QTCoreImage101

QTCoreVideo102

QTCoreVideo201

QTCoreVideo202

QTCoreVideo301

Declared In

`CVDisplayLink.h`

CVDisplayLinkTranslateTime

Translates the time in the display link's time base from one representation to another.

```
CVReturn CVDisplayLinkTranslateTime (
   CVDisplayLinkRef displayLink,
   const CVTimeStamp *inTime,
   CVTimeStamp *outTime
);
```

Parameters

`displayLink`
: The display link whose time base should be used to do the translation.

`inTime`
: A pointer to a `CVTimeStamp` structure containing the source time to translate.

`outTime`
: A pointer to a `CVTimeStamp` structure into which the target time is written. Before calling, you must set the version field (currently `0`) to indicate which version of the structure you want. You should also set the `flags` field to specify which representations to translate to.

Return Value

A Core Video result code. See "Result Codes" (page 358) for possible values.

Discussion

Note that the display link has to be running for this call to succeed.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVDisplayLink.h`

CVGetCurrentHostTime

Retrieves the current value of the host time base.

```
uint64_t CVGetCurrentHostTime
```

Return Value

The current host time.

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVHostTime.h`

CVGetHostClockFrequency

Retrieve the frequency of the host time base.

```
double CVGetHostClockFrequency
```

Return Value
The current host frequency.

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVHostTime.h`

CVGetHostClockMinimumTimeDelta

Retrieve the smallest possible increment in the host time base.

```
uint32_t CVGetHostClockMinimumTimeDelta
```

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVHostTime.h`

CVImageBufferGetCleanRect

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

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

Parameters

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

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

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVImageBuffer.h`

CVImageBufferGetColorSpace

Returns the color space of a Core Video image buffer.

```
CGColorSpaceRef CVImageBufferGetColorSpace (
   CVImageBufferRef imageBuffer
);
```

Parameters

imageBuffer
The image buffer that you want to retrieve the color space from.

Return Value
The color space of the buffer. Returns `NULL` if called with either a non-`CVImageBufferRef` type or `NULL`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVImageBuffer.h`

CVImageBufferGetDisplaySize

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

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

Parameters

imageBuffer

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

Return Value

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

Discussion

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

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVImageBuffer.h`

CVImageBufferGetEncodedSize

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

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

Parameters

imageBuffer

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

Return Value

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

Discussion

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

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVImageBuffer.h`

CVOpenGLBufferAttach

Attaches an OpenGL context to a Core Video OpenGL buffer.

```
CVReturn CVOpenGLBufferAttach (
   CVOpenGLBufferRef openGLBuffer,
   CGLContextObj cglContext,
   GLenum face,
   GLint level,
   GLint screen
);
```

Parameters

openGLBuffer
: The buffer you want to attach an OpenGL context to.

cglContext
: The OpenGL context you want to attach.

face
: The OpenGL face enumeration (`0` for noncube maps.)

level
: The mipmap level for drawing in the OpenGL context. This value cannot exceed the maximum mipmap level for this buffer.

screen
: The virtual screen number you want to use for this context.

Return Value

A Core Video result code. See "Result Codes" (page 358) for possible values.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVOpenGLBuffer.h`

CVOpenGLBufferCreate

Create a new Core Video OpenGL buffer that can be used for OpenGL rendering purposes

```
CVReturn CVOpenGLBufferCreate (
   CFAllocatorRef allocator,
   size_t width,
   size_t height,
   CFDictionaryRef attributes,
   CVOpenGLBufferRef *bufferOut
);
```

Parameters

allocator
: The allocator to use to create the Core Video OpenGL buffer. Pass `NULL` to specify the default allocator.

width
: The width of the buffer in pixels.

height
: The height of the buffer in pixels.

attributes

A Core Foundation dictionary containing other desired attributes of the buffer (texture target, internal format, max mipmap level, etc.). May be `NULL`. The following attribute values are assumed if you do not explicitly define them:

- `kCVOpenGLBufferTarget` = `GL_TEXTURE_RECTANGLE_EXT`
- `kCVOpenGLBufferInternalFormat` = `GL_RGBA`
- `kCVOpenGLBufferMaximumMipmapLevel` = 0

bufferOut

On return, `bufferOut` points to the newly created OpenGL buffer.

Return Value

A Core Video result code. See "Result Codes" (page 358) for possible values.

Discussion

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVOpenGLBuffer.h`

CVOpenGLBufferGetAttributes

Obtains the attributes of a Core Video OpenGL buffer.

```
CFDictionaryRef CVOpenGLBufferGetAttributes (
   CVOpenGLBufferRef openGLBuffer
);
```

Parameters

openGLBuffer

The OpenGL buffer whose attributes you want to obtain.

Return Value

A Core Foundation dictionary containing the OpenGL buffer attributes, or `NULL` if no attributes exist.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVOpenGLBuffer.h`

CVOpenGLBufferGetTypeID

Obtains the Core Foundation type ID for the OpenGL buffer type.

```
CFTypeID CVOpenGLBufferGetTypeID (
   void
);
```

Return Value

The Core Foundation ID for this data type.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVOpenGLBuffer.h`

CVOpenGLBufferPoolCreate

Creates a new OpenGL buffer pool.

```
CVReturn CVOpenGLBufferPoolCreate (
   CFAllocatorRef allocator,
   CFDictionaryRef poolAttributes,
   CFDictionaryRef openGLBufferAttributes,
   CVOpenGLBufferPoolRef *poolOut
);
```

Parameters

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

poolAttributes
    A Core Foundation dictionary containing the attributes to be used for the pool itself.

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

poolOut
    On return, `poolOut` points to the new OpenGL buffer pool.

Return Value
A Core Video result code. See "Result Codes" (page 358) for possible values.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVOpenGLBufferPool.h`

CVOpenGLBufferPoolCreateOpenGLBuffer

Creates a new OpenGL buffer from an OpenGL buffer pool.

```
CVReturn CVOpenGLBufferPoolCreateOpenGLBuffer (
   CFAllocatorRef allocator,
   CVOpenGLBufferPoolRef openGLBufferPool,
   CVOpenGLBufferRef *openGLBufferOut
);
```

Parameters

allocator
    The allocator to use for creating the buffer. May be `NULL` to specify the default allocator.

openGLBufferPool
    The OpenGL buffer pool that should create the new OpenGL buffer.

openGLBufferOut

On return, `OpenGLBufferOut` points to the new OpenGL buffer.

Return Value

A Core Video result code. See "Result Codes" (page 358) for possible values.

Discussion

The function creates a new OpenGL buffer using the OpenGL buffer attributes specified in the `CVOpenGLBufferPoolCreate` (page 304) call. This buffer has default attachments as specified in the `openGLBufferAttributes` parameter of `CVOpenGLBufferPoolCreate` (page 304) (using either the `kCVBufferPropagatedAttachmentsKey` or `kCVBufferNonPropagatedAttachmentsKey` attributes).

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVOpenGLBufferPool.h`

CVOpenGLBufferPoolGetAttributes

Returns the pool attributes dictionary for an Open GL buffer pool.

```
CFDictionaryRef CVOpenGLBufferPoolGetAttributes (
   CVOpenGLBufferPoolRef pool
);
```

Parameters

pool

The OpenGL buffer pool to retrieve the attributes from.

Return Value

The buffer-pool attributes Core Foundation dictionary, or `NULL` on failure.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVOpenGLBufferPool.h`

CVOpenGLBufferPoolGetOpenGLBufferAttributes

Returns the attributes of OpenGL buffers that will be created from a buffer pool.

```
CFDictionaryRef CVOpenGLBufferPoolGetOpenGLBufferAttributes (
   CVOpenGLBufferPoolRef pool
);
```

Parameters

pool

The OpenGL buffer pool to retrieve the attributes from.

Return Value

The OpenGL buffer attributes Core Foundation dictionary, or `NULL` on failure.

Discussion
You can use this function to obtain information about the OpenGL buffers that will be created from the buffer pool.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVOpenGLBufferPool.h`

CVOpenGLBufferPoolGetTypeID

Obtains the Core Foundation ID for the OpenGL buffer pool type.

```
CFTypeID CVOpenGLBufferPoolGetTypeID (
   void
);
```

Return Value
The Core Foundation ID for this data type.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVOpenGLBufferPool.h`

CVOpenGLBufferPoolRelease

Releases an OpenGL buffer pool.

```
void CVOpenGLBufferPoolRelease (
   CVOpenGLBufferPoolRef openGLBufferPool
);
```

Parameters
openGLBufferPool
The OpenGL buffer pool that you want to release.

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVOpenGLBufferPool.h`

CVOpenGLBufferPoolRetain

Retains an OpenGL buffer pool.

```
CVOpenGLBufferPoolRef CVOpenGLBufferPoolRetain (
   CVOpenGLBufferPoolRef openGLBufferPool
);
```

Parameters

openGLBufferPool

The OpenGL buffer pool that you want to retain.

Return Value

For convenience, the same buffer pool object you wanted to retain.

Discussion

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

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVOpenGLBufferPool.h`

CVOpenGLBufferRelease

Releases a Core Video OpenGL buffer.

```
void CVOpenGLBufferRelease (
   CVOpenGLBufferRef buffer
);
```

Parameters

buffer

The OpenGL buffer that you want to release.

Discussion

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

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVOpenGLBuffer.h`

CVOpenGLBufferRetain

Retains a Core Video OpenGL buffer.

```
CVOpenGLBufferRef CVOpenGLBufferRetain (
   CVOpenGLBufferRef buffer
);
```

Parameters

buffer

The OpenGL Buffer that you want to retain.

Return Value

For convenience, the OpenGL buffer that was retained.

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVOpenGLBuffer.h`

CVOpenGLTextureCacheCreate

Creates an OpenGL texture cache.

```
CVReturn CVOpenGLTextureCacheCreate (
   CFAllocatorRef allocator,
   CFDictionaryRef cacheAttributes,
   CGLContextObj cglContext,
   CGLPixelFormatObj cglPixelFormat,
   CFDictionaryRef textureAttributes,
   CVOpenGLTextureCacheRef *cacheOut
);
```

Parameters

`allocator`
The allocator to use for allocating the cache. Pass `NULL` to specify the default allocator.

`cacheAttributes`
A Core Foundation dictionary containing the attributes of the cache itself. Pass `NULL` to specify no attributes.

`cglContext`
The OpenGL context into which the texture objects will be created.

`cglPixelFormat`
The OpenGL pixel format used to create the OpenGL context specified in `cglContext`.

`textureAttributes`
A Core Foundation dictionary containing the attributes to be used for creating the OpenGL texture objects. Pass `NULL` to specify no attributes.

`cacheOut`
On return, `cacheOut` points to the newly created texture cache.

Return Value
A Core Video result code. See "Result Codes" (page 358) for possible values.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVOpenGLTextureCache.h`

CVOpenGLTextureCacheCreateTextureFromImage

Creates an OpenGL texture object from an existing image buffer.

```
CVReturn CVOpenGLTextureCacheCreateTextureFromImage (
   CFAllocatorRef allocator,
   CVOpenGLTextureCacheRef textureCache,
   CVImageBufferRef sourceImage,
   CFDictionaryRef attributes,
   CVOpenGLTextureRef *textureOut
);
```

Parameters

allocator
: The allocator to use for allocating the OpenGL texture object. May be `NULL` to specify the default allocator.

textureCache
: The OpenGL texture cache to be used to manage the texture.

sourceImage
: The image buffer from which you want to create an OpenGL texture.

attributes
: The desired buffer attributes for the OpenGL texture.

textureOut
: On return, `textureOut` points to the newly created texture object.

Return Value
A Core Video result code. See "Result Codes" (page 358) for possible values.

Discussion
This function copies all image buffer attachments designated as propagatable to the newly-created texture.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVOpenGLTextureCache.h`

CVOpenGLTextureCacheFlush

Flushes the OpenGL texture cache.

```
void CVOpenGLTextureCacheFlush (
   CVOpenGLTextureCacheRef textureCache,
   CVOptionFlags options
);
```

Parameters

textureCache
: The texture cache to flush.

options
: Currently unused; pass `0`.

Return Value
A Core Video result code. See "Result Codes" (page 358) for possible values.

Discussion
You must call this function periodically to allow the texture cache to perform housekeeping operations.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVOpenGLTextureCache.h`

CVOpenGLTextureCacheGetTypeID

Returns the Core Foundation ID of the texture cache type.

```
CFTypeID CVOpenGLTextureCacheGetTypeID (
   void
);
```

Return Value
The Core Foundation ID for this type.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVOpenGLTextureCache.h`

CVOpenGLTextureCacheRelease

Releases an OpenGL texture cache.

```
void CVOpenGLTextureCacheRelease (
   CVOpenGLTextureCacheRef textureCache
);
```

Parameters

`textureCache`
    The OpenGL texture cache that you want to release.

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVOpenGLTextureCache.h`

CVOpenGLTextureCacheRetain

Retains an OpenGL texture cache.

```
CVOpenGLTextureCacheRef CVOpenGLTextureCacheRetain (
   CVOpenGLTextureCacheRef textureCache
);
```

Parameters

textureCache

The OpenGL texture cache that you want to retain.

Return Value

For convenience, the return value is the buffer you wanted to retain.

Discussion

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

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVOpenGLTextureCache.h`

CVOpenGLTextureGetCleanTexCoords

Returns the texture coordinates for the part of the image that should be displayed.

```
void CVOpenGLTextureGetCleanTexCoords (
   CVOpenGLTextureRef image,
   GLfloat lowerLeft[2],
   GLfloat lowerRight[2],
   GLfloat upperRight[2],
   GLfloat upperLeft[2]
);
```

Parameters

image

The Core Video OpenGL texture whose clean tex coordinates you want to obtain.

lowerLeft

On return, the `GLFloat` array hold the *s* and *t* texture coordinates of the lower-left corner of the image.

lowerRight

On return, the `GLFloat` array hold the *s* and *t* texture coordinates of the lower-right corner of the image.

upperRight

On return, the `GLFloat` array hold the *s* and *t* texture coordinates of the upper-right corner of the image.

upperLeft

On return, the `GLFloat` array hold the *s* and *t* texture coordinates of the upper-left corner of the image.

Discussion

This function automatically takes into account whether or not the texture is flipped.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

LiveVideoMixer3

QTCoreVideo102

QTCoreVideo201

QTCoreVideo301

QTQuartzPlayer

Declared In

`CVOpenGLTexture.h`

CVOpenGLTextureGetName

Returns the texture target name of a CoreVideo OpenGL texture.

```
GLuint CVOpenGLTextureGetName (
   CVOpenGLTextureRef image
);
```

Parameters

`image`

The Core Video OpenGL texture whose texture target name you want to obtain.

Return Value

The target name of the texture.

Discussion

See the OpenGL specification for more information about texture targets.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

LiveVideoMixer2

LiveVideoMixer3

QTCoreVideo102

QTCoreVideo201

QTCoreVideo301

Declared In

`CVOpenGLTexture.h`

CVOpenGLTextureGetTarget

Returns the texture target (for example, `GL_TEXTURE_2D`) of an OpenGL texture.

```
GLenum CVOpenGLTextureGetTarget (
   CVOpenGLTextureRef image
);
```

Parameters

`image`

The Core Video OpenGL texture whose target you want to obtain.

Return Value
The OpenGL texture target.

Discussion
See the OpenGL specification for more information about texture targets.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
LiveVideoMixer2
LiveVideoMixer3
QTCoreVideo102
QTCoreVideo201
QTCoreVideo301

Declared In
`CVOpenGLTexture.h`

CVOpenGLTextureGetTypeID

Obtains the Core Foundation ID for the Core Video OpenGL texture type.

```
CFTypeID CVOpenGLTextureGetTypeID (
   void
);
```

Return Value
The Core Foundation ID for this type.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTQuartzPlayer

Declared In
`CVOpenGLTexture.h`

CVOpenGLTextureIsFlipped

Determines whether or not an OpenGL texture is flipped vertically.

```
Boolean CVOpenGLTextureIsFlipped (
   CVOpenGLTextureRef image
);
```

Parameters
`image`
The Core Video OpenGL texture whose orientation you want to determine.

Return Value
Returns `true` if (0,0) in the texture is in the upper-left corner, `false` if (0,0) is in the lower left corner.

Discussion
Quartz assumes a lower-left origin.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVOpenGLTexture.h`

CVOpenGLTextureRelease

Releases a Core Video OpenGL texture.

```
void CVOpenGLTextureRelease (
   CVOpenGLTextureRef texture
);
```

Parameters

`texture`
: The Core Video OpenGL texture that you want to release.

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

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CIVideoDemoGL

LiveVideoMixer3

QTCoreImage101

QTCoreVideo102

QTCoreVideo201

Declared In
`CVOpenGLTexture.h`

CVOpenGLTextureRetain

Retains a Core Video OpenGL texture.

```
CVOpenGLTextureRef CVOpenGLTextureRetain (
   CVOpenGLTextureRef texture
);
```

Parameters

`texture`
: The Core Video OpenGL texture that you want to retain.

Return Value
For convenience, the Core Video OpenGL texture you want to retain.

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVOpenGLTexture.h`

CVPixelBufferCreate

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

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

Parameters

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

width
Width of the pixel buffer, in pixels.

height
Height of the pixel buffer, in pixels.

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

pixelBufferAttributes
A dictionary with additonal attributes for a pixel buffer. This parameter is optional. See "Pixel Buffer Attribute Keys" (page 351) for more details.

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

Return Value
A Core Video result code. See "Result Codes" (page 358) for possible values.

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

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferCreateResolvedAttributesDictionary

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

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

Parameters

`allocator`

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

`attributes`

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

`resolvedDictionaryOut`

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

Return Value

A Core Video result code. See "Result Codes" (page 358) for possible values.

Discussion

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

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

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVPixelBuffer.h`

CVPixelBufferCreateWithBytes

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

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

Parameters

`allocator`

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

width
: Width of the pixel buffer, in pixels.

height
: Height of the pixel buffer, in pixels.

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

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

bytesPerRow
: Row bytes of the pixel storage memory.

releaseCallback
: The callback function to be called when the pixel buffer is destroyed. This callback allows the owner of the pixels to free the memory. See `CVPixelBufferReleaseBytesCallback` (page 335) for more information.

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

pixelBufferAttributes
: A Core Foundation dictionary with additonal attributes for a a pixel buffer. This parameter is optional. See "Pixel Buffer Attribute Keys" (page 351) for more details.

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

Return Value
A Core Video result code. See "Result Codes" (page 358) for possible values.

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferCreateWithPlanarBytes

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

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

Parameters

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

width
Width of the pixel buffer, in pixels.

height
Height of the pixel buffer, in pixels.

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

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

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

numberOfPlanes
The number of planes.

planeBaseAddress
The array of base addresses for the planes.

planeWidth
The array of plane widths.

planeHeight
The array of plane heights.

planeBytesPerRow
Thje array of plane bytes-per-row values.

releaseCallback
The callback function that gets called when the pixel buffer is destroyed. This callback allows the owner of the pixels to free the memory. See `CVPixelBufferReleaseBytesCallback` (page 335) for more information.

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

pixelBufferAttributes
A dictionary with additonal attributes for a a pixel buffer. This parameter is optional. See "Pixel Buffer Attribute Keys" (page 351) for more details.

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

Return Value
A Core Video result code. See "Result Codes" (page 358) for possible values.

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferFillExtendedPixels

Fills the extended pixels of the pixel buffer.

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

Parameters

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

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetBaseAddress

Returns the base address of the pixel buffer.

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

Parameters

pixelBuffer
The pixel buffer whose base address you want to obtain.

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

Discussion
Retrieving the base address for a pixel buffer requires that the buffer base address be locked via a successful call to `CVPixelBufferLockBaseAddress` (page 326).

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie
MovieVideoChart
OpenGLCaptureToMovie
QTPixelBufferVCToCGImage
Quartz Composer QCTV

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetBaseAddressOfPlane

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

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

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

`planeIndex`
The index of the plane.

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

Discussion
Retrieving the base address for a pixel buffer requires that the buffer base address be locked by a successful call to `CVPixelBufferLockBaseAddress` (page 326).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetBytesPerRow

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

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

Parameters

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

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

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie
MovieVideoChart
OpenGLCaptureToMovie
QTPixelBufferVCToCGImage
Quartz Composer QCTV

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetBytesPerRowOfPlane

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

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

Parameters

pixelBuffer
The pixel buffer containing the plane.

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

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetDataSize

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

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

Parameters

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

Return Value
The data size as specified in the call to `CVPixelBufferCreateWithPlanarBytes` (page 317).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetExtendedPixels

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

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

Parameters

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

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

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

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

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

Discussion

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetHeight

Returns the height of the pixel buffer.

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

Parameters

pixelBuffer

The pixel buffer whose height you want to obtain.

Return Value

The buffer height, in pixels.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CaptureAndCompressIPBMovie

MovieVideoChart

QTPixelBufferVCToCGImage

Declared In

`CVPixelBuffer.h`

CVPixelBufferGetHeightOfPlane

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

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

Parameters

pixelBuffer

The pixel buffer whose plane height you want to obtain.

planeIndex

The index of the plane.

Return Value

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

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVPixelBuffer.h`

CVPixelBufferGetPixelFormatType

Returns the pixel format type of the pixel buffer.

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

Parameters

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

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

Discussion

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTPixelBufferVCToCGImage

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetPlaneCount

Returns number of planes of the pixel buffer.

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

Parameters

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

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetTypeID

Returns the Core Foundation ID of the pixel buffer type.

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

Return Value
The Core Foundation ID for this type.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetWidth

Returns the width of the pixel buffer.

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

Parameters

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

Return Value
The width of the buffer, in pixels.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie
MovieVideoChart
QTPixelBufferVCToCGImage

Declared In
`CVPixelBuffer.h`

CVPixelBufferGetWidthOfPlane

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

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

Parameters

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

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

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferIsPlanar

Determine if the pixel buffer is planar.

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

Parameters

`pixelBuffer`

The pixel buffer to check.

Return Value
Returns `true` if the pixel buffer was created using `CVPixelBufferCreateWithPlanarBytes` (page 317), `false` otherwise.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferLockBaseAddress

Locks the base address of the pixel buffer.

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

Parameters

`pixelBuffer`

The pixel buffer whose base address you want to lock.

`lockFlags`

No options currently defined; pass `0`.

Return Value
A Core Video result code. See "Result Codes" (page 358) for possible values.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie

MovieVideoChart

OpenGLCaptureToMovie

QTPixelBufferVCToCGImage

Quartz Composer QCTV

Declared In
`CVPixelBuffer.h`

CVPixelBufferPoolCreate

Creates a pixel buffer pool.

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

Parameters

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

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

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

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

Return Value
A Core Video result code. See "Result Codes" (page 358) for possible values.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
OpenGLCaptureToMovie

Quartz Composer QCTV

Declared In
`CVPixelBufferPool.h`

CVPixelBufferPoolCreatePixelBuffer

Creates a pixel buffer from a pixel buffer pool.

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

Parameters

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

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

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

Return Value
A Core Video result code. See "Result Codes" (page 358) for possible values.

Discussion
This function creates a new pixel buffer using the pixel buffer attributes specifed during pool creation. This buffer has default attachments as specified in the `pixelBufferAttributes` parameter of `CVPixelBufferPoolCreate` (page 327) (using either the `kCVBufferPropagatedAttachmentsKey` or `kCVBufferNonPropagatedAttachmentsKey` attributes).

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
OpenGLCaptureToMovie
Quartz Composer QCTV

Declared In
`CVPixelBufferPool.h`

CVPixelBufferPoolGetAttributes

Returns the pool attributes dictionary for a pixel buffer pool.

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

Parameters

pool
The pixel buffer pool to retrieve the attributes from.

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

Discussion

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVPixelBufferPool.h`

CVPixelBufferPoolGetPixelBufferAttributes

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

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

Parameters

pool

The pixel buffer pool to retrieve the attributes from.

Return Value

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

Discussion

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

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVPixelBufferPool.h`

CVPixelBufferPoolGetTypeID

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

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

Return Value

The Core Foundation ID for this type.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVPixelBufferPool.h`

CVPixelBufferPoolRelease

Releases a pixel buffer pool.

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

Parameters

pixelBufferPool

The pixel buffer pool that you want to release.

Discussion

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

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

OpenGLCaptureToMovie

Quartz Composer QCTV

Declared In

`CVPixelBufferPool.h`

CVPixelBufferPoolRetain

Retains a pixel buffer pool.

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

Parameters

`buffer`

The pixel buffer pool that you want to retain.

Return Value

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

Discussion

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

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVPixelBufferPool.h`

CVPixelBufferRelease

Releases a pixel buffer.

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

Parameters

`buffer`

The pixel buffer that you want to release.

Discussion

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

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

OpenGLCaptureToMovie

QTCoreVideo103

QTCoreVideo202

QTPixelBufferVCToCGImage

Declared In
`CVPixelBuffer.h`

CVPixelBufferRetain

Retains a pixel buffer.

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

Parameters

buffer

The pixel buffer that you want to retain.

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

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferUnlockBaseAddress

Unlocks the base address of the pixel buffer.

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

Parameters

pixelBuffer

The pixel buffer whose base address you want to unlock.

unlockFlags

No options currently defined; pass `0`.

Return Value
A Core Video result code. See "Result Codes" (page 358) for possible values.

Discussion

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie
MovieVideoChart
QTCoreVideo202

QTPixelBufferVCToCGImage
Quartz Composer QCTV

Declared In
`CVPixelBuffer.h`

CVPixelFormatDescriptionArrayCreateWithAllPixelFormatTypes

Returns all the pixel format descriptions known to Core Video.

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

Parameters

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

Return Value
An array of Core Foundation dictionaries, each containing a pixel format description. See "Pixel Format Description Keys" (page 353) for a list of keys relevant to the format description.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVPixelFormatDescription.h`

CVPixelFormatDescriptionCreateWithPixelFormatType

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

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

Parameters

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

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

Return Value
A Core Foundation dictionary containing the pixel format description. See "Pixel Format Description Keys" (page 353) for a list of keys relevant to the format description.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVPixelFormatDescription.h`

CVPixelFormatDescriptionRegisterDescriptionWithPixelFormatType

Registers a pixel format description with Core Video.

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

Parameters

description

A Core Foundation dictionary containing the pixel format description. See "Pixel Format Description Keys" (page 353) for a list of required and optional keys.

pixelFormat

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

Discussion

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

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVPixelFormatDescription.h`

Callbacks

CVDisplayLinkOutputCallback

Defines a pointer to a display link output callback function, which is called whenever the display link wants the application to output a frame.

```
typedef CVReturn (*CVDisplayLinkOutputCallback)(
   CVDisplayLinkRef displayLink,
   const CVTimeStamp *inNow,
   const CVTimeStamp *inOutputTime,
   CVOptionFlags flagsIn,
   CVOptionFlags *flagsOut,
   void *displayLinkContext
   );
```

You would declare a display link output callback function named `MyDisplayLinkCallback` like this:

```
CVReturn MyDisplayLinkCallback (
   CVDisplayLinkRef displayLink,
   const CVTimeStamp *inNow,
   const CVTimeStamp *inOutputTime,
   CVOptionFlags flagsIn,
   CVOptionFlags *flagsOut,
   void *displayLinkContext
   );
```

Parameters

displayLink
: The display link requesting the frame.

inNow
: A pointer to the current time.

inOutputTime
: A pointer to the time that the frame will be displayed.

flagsIn
: Currently unused. Pass 0.

flagsOut
: Currently unused. Pass 0.

displayLinkContext
: A pointer to application-defined data. This is the pointer you passed into the `CVDisplayLinkSetOutputCallback` (page 296) function when registering your callback.

Discussion

For a given display link, you must register a display link output callback using `CVDisplayLinkSetOutputCallback` (page 296) so that you can process and output the requested frame.

You callback must retrieve the frame with the timestamp specified by the (`inOutputTime` parameter, manipulate it if desired (for example, apply color correction or map into onto a surface), and then output it to the display.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVDisplayLink.h`

CVFillExtendedPixelsCallBack

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

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

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

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

Parameters

pixelBuffer
: The pixel buffer to be padded.

refCon

A pointer to application-defined data. This is the same value you stored in the `CVFillExtendedPixelsCallbackData` (page 337) structure.

Return Value

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

Discussion

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

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVPixelFormatDescription.h`

CVPixelBufferReleaseBytesCallback

Defines a pointer to a pixel buffer release callback function, which is called when a pixel buffer created by `CVPixelBufferCreateWithBytes` (page 316) is released.

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

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

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

Parameters

releaseRefCon

A pointer to application-defined data. This pointer is the same as that passed in the `releaseRefCon` parameter of `CVPixelBufferCreateWithBytes` (page 316).

baseAddress

A pointer to the base address of the memory holding the pixels. This pointer is the same as that passed in the `baseAddress` parameter of `CVPixelBufferCreateWithBytes` (page 316).

Discussion

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

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVPixelBuffer.h`

CVPixelBufferReleasePlanarBytesCallback

Defines a pointer to a pixel buffer release callback function, which is called when a pixel buffer created by `CVPixelBufferCreateWithPlanarBytes` (page 317) is released.

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

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

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

Parameters

`releaseRefCon`
: A pointer to application-defined data. This pointer is the same as that passed in the `releaseRefCon` parameter of `CVPixelBufferCreateWithPlanarBytes` (page 317).

`dataPtr`
: A pointer to a plane descriptor block. This is the same pointer you passed to `CVPixelBufferCreateWithPlanarBytes` (page 317) in the `dataPtr` parameter.

`dataSize`
: The size value you passed to `CVPixelBufferCreateWithPlanarBytes` (page 317) in the `dataSize` parameter.

`numberOfPlanes`
: The number of planes value you passed to `CVPixelBufferCreateWithPlanarBytes` (page 317) in the `numberOfPlanes` parameter.

`planeAddresses`
: A pointer to the base plane address you passed to `CVPixelBufferCreateWithPlanarBytes` (page 317) in the `basePlaneAddress` parameter.

Discussion

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

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVPixelBuffer.h`

Data Types

CVBufferRef

Defines the base type for all Core Video buffers.

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

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVBuffer.h`

CVDisplayLinkRef

Defines a display link.

```
typedef struct __CVDisplayLink *CVDisplayLinkRef;
```

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVDisplayLink.h`

CVFillExtendedPixelsCallbackData

Holds information describing a custom extended pixel fill algorithm.

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

Fields

`version`
The version of this fill algorithm.

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

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

Discussion
You must fill out this structure and store it as part of your pixel format description Core Foundation dictionary (key: `kCVPixelFormatFillExtendedPixelsCallback`, type: `CFData`). However, if your custom pixel format never needs the functionality of `CVPixelBufferFillExtendedPixels` (page 319), you don't need to add this key or implement the associated callback.

For more information about defining a custom pixel format, see "Pixel Format Description Keys" (page 353).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVPixelFormatDescription.h`

CVImageBufferRef

Defines a Core Video image buffer.

```
typedef CVBufferRef CVImageBufferRef;
```

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVImageBuffer.h`

CVOptionFlags

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

```
typedef uint64_t CVOptionFlags;
```

Discussion
No flags are currently defined.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVBase.h`

CVOpenGLBufferRef

Defines a Core Video OpenGL buffer.

```
typedef CVImageBufferRef CVOpenGLBufferRef;
```

Discussion
The Core Video OpenGL buffer (type `CVOpenGLBufferRef` is a wrapper around the standard OpenGL pbuffer.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVOpenGLBuffer.h`

CVOpenGLBufferPoolRef

Defines an OpenGL buffer pool.

```
typedef struct _CVOpenGLBufferPool *CVOpenGLBufferPoolRef;
```

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVOpenGLBufferPool.h`

CVOpenGLTextureRef

Defines an OpenGL texture-based image buffer.

```
typedef CVImageBufferRef CVOpenGLTextureRef;
```

Discussion
The Core Video OpenGL texture (type `CVOpenGLTextureRef` is a wrapper around the standard OpenGL texture.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVOpenGLTexture.h`

CVOpenGLTextureCacheRef

Defines a CoreVideo OpenGL texture cache.

```
typedef struct __CVOpenGLTextureCache *CVOpenGLTextureCacheRef;
```

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVOpenGLTextureCache.h`

CVPixelBufferRef

Defines a Core Video pixel buffer.

```
typedef CVImageBufferRef CVPixelBufferRef;
```

Discussion
The pixel buffer stores an image in main memory.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferPoolRef

Defines a pixel buffer pool.

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVPixelBufferPool.h`

CVReturn

Defines the return error code for Core Video functions.

```
typedef int32_t CVReturn;
```

Discussion
See "Result Codes" (page 358) for possible values.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVReturn.h`

CVSMPTETime

A structure for holding a SMPTE time.

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

Fields

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

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

`counter`
: The total number of messages received.

`type`
: The kind of SMPTE time type. See "SMPTE Time Types" (page 357) for a list of possible values.

`flags`
: A set of flags that indicate the SMPTE state. See "SMPTE State Flags" (page 356) for possible values.

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

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

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

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVBase.h`

CVTime

A structure for reporting Core Video time values.

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

Fields

`timeValue`
: The time value.

`timeScale`
: The time scale for this value.

`flags`
: Flags associated with the `CVTime` value. See "CVTime Constants" (page 344) for possible values. If `kCVTimeIsIndefinite` is set, you should not use any of the other fields in this structure.

Discussion

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

Availability

Available in Mac OS X v10.3 and later.

Declared In

`CVBase.h`

CVTimeStamp

A structure for defining a display timestamp.

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

Fields

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

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

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

`hostTime`
: The host root time base time.

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

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

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

`flags`
: A bit field containing additional information about the timestamp. See "CVTimeStamp Flags" (page 345) for a list of possible values. .

`reserved`
: Reserved. Do not use.

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

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CVBase.h`

Constants

CVBuffer Attachment Keys

Specify an attachment type for a Core Video buffer.

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

Constants

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

  Available in Mac OS X v10.3 and later.

  Declared in `CVBuffer.h`.

`kCVBufferTimeValueKey`
: The actual time value associated with the movie.

  Available in Mac OS X v10.3 and later.

  Declared in `CVBuffer.h`.

`kCVBufferTimeScaleKey`
: The time scale associated with the movie.

  Available in Mac OS X v10.3 and later.

  Declared in `CVBuffer.h`.

CVBuffer Attachment Modes

Specify the propagation mode of a Core Video buffer attachment.

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

Constants

`kCVAttachmentMode_ShouldNotPropagate`
: Do not propagate this attachment.

  Available in Mac OS X v10.3 and later.

  Declared in `CVBuffer.h`.

`kCVAttachmentMode_ShouldPropagate`
: Copy this attachment when using the `CVBufferPropagateAttachments` (page 284) function. For example, in most cases, you would want to propagate an attachment bearing a timestamp to each successive buffer.

  Available in Mac OS X v10.3 and later.

  Declared in `CVBuffer.h`.

Discussion

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

CVBuffer Attribute Keys

Specify attributes associated with Core Video buffers.

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

Constants

`kCVBufferPropagatedAttachmentsKey`
: Attachments that should be copied when using the `CVBufferPropagateAttachments` (page 284) function (type `CFDictionary`, containing a list of attachments as key-value pairs).

  Available in Mac OS X v10.3 and later.

  Declared in `CVBuffer.h`.

`kCVBufferNonPropagatedAttachmentsKey`
: Attachments that should not be copied when using the `CVBufferPropagateAttachments` (page 284) function (type `CFDictionary`, containing a list of attachments as key-value pairs).

  Available in Mac OS X v10.3 and later.

  Declared in `CVBuffer.h`.

Discussion

These attributes let you set multiple attachments at the time of buffer creation, rather than having to call `CVBufferSetAttachment` (page 287) for each attachment.

CVTime Constants

Specify flags for the `CVTime` structure.

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

Constants

`kCVTimeIsIndefinite`

The time value is unknown.

Available in Mac OS X v10.3 and later.

Declared in `CVBase.h`.

CVTime Values

Indicate specific `CVTime` values.

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

Constants

`kCVZeroTime`

Zero time or duration. For example, `CVDisplayLinkGetOutputVideoLatency` (page 292) returns `kCVZeroTime` for zero video latency.

Available in Mac OS X v10.3 and later.

Declared in `CVBase.h`.

`kCVIndefiniteTime`

An unknown or indefinite time. For example, `CVDisplayLinkGetNominalOutputVideoRefreshPeriod` (page 292) returns `kCVIndefiniteTime` if the display link specified is not valid.

Available in Mac OS X v10.3 and later.

Declared in `CVBase.h`.

CVTimeStamp Flags

Specify flags for the `CVTimeStamp` structure.

```
enum
{
    kCVTimeStampVideoTimeValid              = (1L << 0),
    kCVTimeStampHostTimeValid               = (1L << 1),
    kCVTimeStampSMPTETimeValid              = (1L << 2),
    kCVTimeStampVideoRefreshPeriodValid     = (1L << 3),
    kCVTimeStampRateScalarValid             = (1L << 4),
    kCVTimeStampTopField                    = (1L << 16),
    kCVTimeStampBottomField                 = (1L << 17)
};
enum
{
    kCVTimeStampVideoHostTimeValid   =
            (kCVTimeStampVideoTimeValid | kCVTimeStampHostTimeValid),
    kCVTimeStampIsInterlaced             =
            (kCVTimeStampTopField | kCVTimeStampBottomField)
};
```

Constants

`kCVTimeStampVideoTimeValid`

The value in the video time field is valid.

Available in Mac OS X v10.3 and later.

Declared in `CVBase.h`.

`kCVTimeStampHostTimeValid`

The value in the host time field is valid.

Available in Mac OS X v10.3 and later.

Declared in `CVBase.h`.

`kCVTimeStampSMPTETimeValid`

The value in the SMPTE time field is valid.

Available in Mac OS X v10.3 and later.

Declared in `CVBase.h`.

`kCVTimeStampVideoRefreshPeriodValid`

The value in the video refresh period field is valid.

Available in Mac OS X v10.3 and later.

Declared in `CVBase.h`.

`kCVTimeStampRateScalarValid`

The value in the rate scalar field is valid.

Available in Mac OS X v10.3 and later.

Declared in `CVBase.h`.

`kCVTimeStampTopField`

The timestamp represents the top lines of an interlaced image.

Available in Mac OS X v10.3 and later.

Declared in `CVBase.h`.

`kCVTimeStampBottomField`

The timestamp represents the bottom lines of an interlaced image.

Available in Mac OS X v10.3 and later.

Declared in `CVBase.h`.

`kCVTimeStampVideoHostTimeValid`
A convenience constant indicating that both the video time and host time fields are valid.

Available in Mac OS X v10.3 and later.

Declared in `CVBase.h`.

`kCVTimeStampIsInterlaced`
A convenience constant indicating that the timestamp is for an interlaced image.

Available in Mac OS X v10.3 and later.

Declared in `CVBase.h`.

Discussion

These flags indicate which fields in the `CVTimeStamp` (page 342) structure contain valid information.

Image Buffer Attachment Keys

Specify attachment types associated with image buffers.

```
const CFStringRef   kCVImageBufferCGColorSpaceKey;
const CFStringRef   kCVImageBufferGammaLevelKey;
const CFStringRef   kCVImageBufferCleanApertureKey;
const CFStringRef   kCVImageBufferPreferredCleanApertureKey;
const CFStringRef   kCVImageBufferCleanApertureWidthKey;
const CFStringRef   kCVImageBufferCleanApertureHeightKey;
const CFStringRef   kCVImageBufferCleanApertureHorizontalOffsetKey;
const CFStringRef   kCVImageBufferCleanApertureVerticalOffsetKey;
const CFStringRef   kCVImageBufferFieldCountKey;
const CFStringRef   kCVImageBufferFieldDetailKey;
const CFStringRef   kCVImageBufferFieldDetailTemporalTopFirst;
const CFStringRef   kCVImageBufferFieldDetailTemporalBottomFirst;
const CFStringRef   kCVImageBufferFieldDetailSpatialFirstLineEarly;
const CFStringRef   kCVImageBufferFieldDetailSpatialFirstLineLate;
const CFStringRef   kCVImageBufferPixelAspectRatioKey;
const CFStringRef   kCVImageBufferPixelAspectRatioHorizontalSpacingKey;
const CFStringRef   kCVImageBufferPixelAspectRatioVerticalSpacingKey;
const CFStringRef   kCVImageBufferDisplayDimensionsKey;
const CFStringRef   kCVImageBufferDisplayWidthKey;
const CFStringRef   kCVImageBufferDisplayHeightKey;
const CFStringRef   kCVImageBufferYCbCrMatrixKey;
const CFStringRef   kCVImageBufferYCbCrMatrix_ITU_R_709_2;
const CFStringRef   kCVImageBufferYCbCrMatrix_ITU_R_601_4;
const CFStringRef   kCVImageBufferYCbCrMatrix_SMPTE_240M_1995;
```

Constants

`kCVImageBufferCGColorSpaceKey`
The color space for the buffer (type `CGColorSpaceRef`).

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferGammaLevelKey`
The gamma level for this buffer (type `CFNumber`).

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferCleanApertureKey`
The clean aperture for the buffer (type `CFDictionary` , containing the clean aperture width, height, and horizontal and vertical offset key-value pairs).

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferPreferredCleanApertureKey`
The preferred clean aperture for the buffer (type `CFDictionary` , containing the clean aperture width, height, and horizontal and vertical offset key-value pairs).

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferCleanApertureWidthKey`
The clean aperture width (type `CFNumber`).

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferCleanApertureHeightKey`
The clean aperture height (type `CFNumber`).

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferCleanApertureHorizontalOffsetKey`
The clean aperture horizontal offset (type `CFNumber`).

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferCleanApertureVerticalOffsetKey`
The clean aperture vertical offset (type `CFNumber`).

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferFieldCountKey`
The field count for the buffer (type `CFNumber`).

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferFieldDetailKey`
Specific information about the field of a video frame in the buffer (type `CFDictionary`, containing the temporal bottom first and top first and spacial first-line-early and first-line-late keys).

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferFieldDetailTemporalTopFirst`
(type `CFString`).

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferFieldDetailTemporalBottomFirst`
(type `CFString`).

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferFieldDetailSpatialFirstLineEarly`
(type `CFString`).

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferFieldDetailSpatialFirstLineLate`
(type `CFString`).

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferPixelAspectRatioKey`
The pixel aspect ratio of the buffer (type `CFDictionary`, containing the horizontal and vertical spacing keys).

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferPixelAspectRatioHorizontalSpacingKey`
The horizontal component of the buffer aspect ratio (type `CFNumber`).

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferPixelAspectRatioVerticalSpacingKey`
The vertical component of the buffer aspect ratio (type `CFNumber`).

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferDisplayDimensionsKey`
The buffer display dimensions (type `CFDictionary` containing the buffer display width and height keys).

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferDisplayWidthKey`
The buffer display width (type `CFNumber`).

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferDisplayHeightKey`
The buffer display height (type `CFNumber`).

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferYCbCrMatrixKey`
The type of conversion matrix used for this buffer when converting from YCbCr to RGB images (type `CFString`). The value for this key should be one of the following constants: `kCVImageBufferYCbCrMatrix_ITU_R_709_2`, `kCVImageBufferYCbCrMatrix_ITU_R_601_4`, or `kCVImageBufferYCbCrMatrix_SMPTE_240M_1995`.

Available in Mac OS X v10.3 and later.

Declared in `CVImageBuffer.h`.

`kCVImageBufferYCbCrMatrix_ITU_R_709_2`
: Specifies the YCbCr to RGB conversion matrix for HDTV digital television (ITU R 709) images.

  Available in Mac OS X v10.3 and later.

  Declared in `CVImageBuffer.h`.

`kCVImageBufferYCbCrMatrix_ITU_R_601_4`
: Specifies the YCbCr to RGB conversion matrix for standard digital television (ITU R 601) images.

  Available in Mac OS X v10.3 and later.

  Declared in `CVImageBuffer.h`.

`kCVImageBufferYCbCrMatrix_SMPTE_240M_1995`
: Specifies the YCbCR to RGB conversion matrix for 1920 x 1135 HDTV (SMPTE 240M 1995).

  Available in Mac OS X v10.3 and later.

  Declared in `CVImageBuffer.h`.

Discussion

Image buffer attachment keys are stored in a Core Foundation dictionary associated with an image buffer. Note that some of these keys are stored in subdictionaries keyed by a higher-level attribute. For example, the `kCVImageBufferDisplayWidthKey` and `kCVImageBufferDisplayHeightKey` attributes are stored in a Core Foundation dictionary keyed to the `kCVImageBufferDisplayDimensionsKey` attribute.

OpenGL Buffer Attribute Keys

Specify attributes of an OpenGL buffer.

```
const CFStringRef kCVOpenGLBufferWidth;
const CFStringRef kCVOpenGLBufferHeight;
const CFStringRef kCVOpenGLBufferTarget;
const CFStringRef kCVOpenGLBufferInternalFormat;
const CFStringRef kCVOpenGLBufferMaximumMipmapLevel;
```

Constants

`kCVOpenGLBufferWidth`
: The width of the buffer.

  Available in Mac OS X v10.3 and later.

  Declared in `CVOpenGLBuffer.h`.

`kCVOpenGLBufferHeight`
: The height of the buffer.

  Available in Mac OS X v10.3 and later.

  Declared in `CVOpenGLBuffer.h`.

`kCVOpenGLBufferTarget`
: The OpenGL target for this buffer.

  Available in Mac OS X v10.3 and later.

  Declared in `CVOpenGLBuffer.h`.

`kCVOpenGLBufferInternalFormat`
: The OpenGL internal format of this buffer.

  Available in Mac OS X v10.3 and later.

  Declared in `CVOpenGLBuffer.h`.

`kCVOpenGLBufferMaximumMipmapLevel`
The maximum mipmap level for this buffer.

Available in Mac OS X v10.3 and later.

Declared in `CVOpenGLBuffer.h`.

OpenGL Buffer Pool Attribute Keys

Specify attributes associated with an OpenGL buffer pool.

```
const CFStringRef kCVOpenGLBufferPoolMinimumBufferCountKey;
const CFStringRef kCVOpenGLBufferPoolMaximumBufferAgeKey;
```

Constants

`kCVOpenGLBufferPoolMinimumBufferCountKey`
Indicates the minimum number of buffers to keep in the pool (type `CFNumber`).

Available in Mac OS X v10.3 and later.

Declared in `CVOpenGLBufferPool.h`.

`kCVOpenGLBufferPoolMaximumBufferAgeKey`
Indicates how long unused buffers should be kept before they are deallocated (type `CFAbsoluteTime`).

Available in Mac OS X v10.3 and later.

Declared in `CVOpenGLBufferPool.h`.

Discussion
You specify these keys in a Core Foundation dictionary when calling functions such as `CVOpenGLBufferPoolCreate` (page 304).

Pixel Buffer Attribute Keys

Specify attributes associated with a pixel buffer.

```
const CFStringRef kCVPixelBufferPixelFormatTypeKey;
 const CFStringRef kCVPixelBufferMemoryAllocatorKey;
 const CFStringRef kCVPixelBufferWidthKey;
 const CFStringRef kCVPixelBufferHeightKey;
 const CFStringRef kCVPixelBufferExtendedPixelsLeftKey;
 const CFStringRef kCVPixelBufferExtendedPixelsTopKey;
 const CFStringRef kCVPixelBufferExtendedPixelsRightKey;
 const CFStringRef kCVPixelBufferExtendedPixelsBottomKey;
 const CFStringRef kCVPixelBufferBytesPerRowAlignmentKey;
 const CFStringRef kCVPixelBufferCGBitmapContextCompatibilityKey;
 const CFStringRef kCVPixelBufferCGImageCompatibilityKey;
 const CFStringRef kCVPixelBufferOpenGLCompatibilityKey;
```

Constants

`kCVPixelBufferPixelFormatTypeKey`
The pixel format for this buffer (type `CFNumber`, or type `CFArray` containing an array of `CFNumber` types (actually type `OSType`)). For a listing of common pixel formats, see the QuickTime Ice Floe Dispatch 20.

Available in Mac OS X v10.3 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferMemoryAllocatorKey`
The allocator used with this buffer (type `CFAllocatorRef`).

Available in Mac OS X v10.3 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferWidthKey`
The width of the pixel buffer (type `CFNumber`).

Available in Mac OS X v10.3 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferHeightKey`
The height of the pixel buffer (type `CFNumber`).

Available in Mac OS X v10.3 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferExtendedPixelsLeftKey`
The number of pixels padding the left of the image (type `CFNumber`).

Available in Mac OS X v10.3 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferExtendedPixelsTopKey`
The number of pixels padding the top of the image (type `CFNumber`).

Available in Mac OS X v10.3 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferExtendedPixelsRightKey`
The number of pixels padding the right of the image (type `CFNumber`).

Available in Mac OS X v10.3 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferExtendedPixelsBottomKey`
The number of pixels padding the bottom of the image (type `CFNumber`).

Available in Mac OS X v10.3 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferBytesPerRowAlignmentKey`
Indicates the number of bytes per row in the pixel buffer (type `CFNumber`).

Available in Mac OS X v10.3 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferCGBitmapContextCompatibilityKey`
Indicates whether the pixel buffer is compatible with Core Graphics bitmap contexts (type `CFBoolean`).

Available in Mac OS X v10.3 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferCGImageCompatibilityKey`
Indicates whether the pixel buffer is compatible with `CGImage` types (type `CFBoolean`).

Available in Mac OS X v10.3 and later.

Declared in `CVPixelBuffer.h`.

`kCVPixelBufferOpenGLCompatibilityKey`
: Indicates whether the pixel buffer is compatible with OpenGL contexts (type `CFBoolean`).

  Available in Mac OS X v10.3 and later.

  Declared in `CVPixelBuffer.h`.

Discussion
You specify these keys in a Core Foundation dictionary when calling functions such as `CVPixelBufferCreate` (page 315).

Pixel Buffer Pool Attribute Keys

Specify attributes associated with a pixel buffer pool.

```
const CFStringRef kCVPixelBufferPoolMinimumBufferCountKey;
const CFStringRef kCVPixelBufferPoolMaximumBufferAgeKey;
```

Constants
`kCVPixelBufferPoolMinimumBufferCountKey`
: The minimum number of buffers allowed in the pixel buffer pool (type `CFNumber`).

  Available in Mac OS X v10.3 and later.

  Declared in `CVPixelBufferPool.h`.

`kCVPixelBufferPoolMaximumBufferAgeKey`
: The maximum allowable age for a buffer in the pixel buffer pool (type `CFAbsoluteTime`).

  Available in Mac OS X v10.3 and later.

  Declared in `CVPixelBufferPool.h`.

Discussion
You specify these keys in a Core Foundation dictionary when calling functions such as `CVPixelBufferPoolCreate` (page 327).

Pixel Format Description Keys

Specify attributes of a pixel format.

```
const CFStringRef kCVPixelFormatName;
const CFStringRef kCVPixelFormatConstant;
const CFStringRef kCVPixelFormatCodecType;
const CFStringRef kCVPixelFormatFourCC;
const CFStringRef kCVPixelFormatPlanes;
const CFStringRef kCVPixelFormatBlockWidth;
const CFStringRef kCVPixelFormatBlockHeight;
const CFStringRef kCVPixelFormatBitsPerBlock;
const CFStringRef kCVPixelFormatBlockHorizontalAlignment;
const CFStringRef kCVPixelFormatBlockVerticalAlignment;
const CFStringRef kCVPixelFormatHorizontalSubsampling;
const CFStringRef kCVPixelFormatVerticalSubsampling;

const CFStringRef kCVPixelFormatOpenGLFormat;
const CFStringRef kCVPixelFormatOpenGLType;
const CFStringRef kCVPixelFormatOpenGLInternalFormat;
const CFStringRef kCVPixelFormatCGBitmapInfo;

const CFStringRef kCVPixelFormatQDCompatibility;
const CFStringRef kCVPixelFormatCGBitmapContextCompatibility;
const CFStringRef kCVPixelFormatCGImageCompatibility;
const CFStringRef kCVPixelFormatOpenGLCompatibility;

const CFStringRef kCVPixelFormatFillExtendedPixelsCallback;
```

Constants

`kCVPixelFormatName`

The name of the pixel format (type `CFString`). This should be the same as the codec name you would use in QuickTime.

Available in Mac OS X v10.3 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatConstant`

The pixel format constant for QuickTime.

Available in Mac OS X v10.3 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatCodecType`

The codec type (type `CFString`). For example, `'2vuy'` or `k422YpCbCr8CodecType`.

Available in Mac OS X v10.3 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatFourCC`

The Microsoft FourCC equivalent code for this pixel format (type `CFString`).

Available in Mac OS X v10.3 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatPlanes`

The number of image planes associated with this format (type `CFNumber`. Each plane may contain a single component or an interleaved set of components. Note that if your pixel format is not planar, you can put the required format keys at the top-level dictionary.

Available in Mac OS X v10.3 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatBlockWidth`

The width, in pixels, of the smallest byte-addressable group of pixels (type `CFNumber`. Used to assist with allocating memory for pixel formats that don't have an integer value for bytes per pixel. Assumed to be 1 if this key is not present. Here are some examples of block widths for standard pixel formats:

- 8-bit luminance only, block width is 1, the bits per block value is 8.
- 16-bit 1555 RGB, block width is 1, the bits per block value is 16.
- 32-bit 8888 ARGB, block width is 1, the bits per block value is 32.
- 2vuy (CbYCrY), block width is 2, the bits per block value is 32.
- 1-bit bitmap, block width is 8, the bits per block value is 8.
- v210, block width is 6, the bits per block value is 128 .

Available in Mac OS X v10.3 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatBlockHeight`
The height, in pixels, of the smallest byte-addressable group of pixels (type `CFNumber`). Assumed to be one if this key is not present.

Available in Mac OS X v10.3 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatBitsPerBlock`
The number of bits per block. For simple pixel formats, this value is the same as the traditional bits-per-pixel value. This key is mandatory in pixel format descriptions. See the description for `kCVPixelFormatBlockWidth` for examples of bits-per-block values.

Available in Mac OS X v10.3 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatBlockHorizontalAlignment`
The horizontal alignment requirements of this format (type `CFNumber`). For example,the alignment for v210 would be '8' here for the horizontal case to match the standard v210 row alignment value of 48. Assumed to be 1 if this key is not present.

Available in Mac OS X v10.3 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatBlockVerticalAlignment`
The vertical alignment requirements of this format (type `CFNumber`). Assumed to be 1 if this key is not present.

Available in Mac OS X v10.3 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatHorizontalSubsampling`
Horizontal subsampling information for this plane (type `CFNumber`). Assumed to be 1 if this key is not present.

Available in Mac OS X v10.3 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatVerticalSubsampling`
Vertical subsampling information for this plane (type `CFNumber`). Assumed to be 1 if this key is not present.

Available in Mac OS X v10.3 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatOpenGLFormat`
The OpenGL format used to describe this image plane (if applicable). See the OpenGL specification for possible values.

Available in Mac OS X v10.3 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatOpenGLType`
The OpenGL type to describe this image plane (if applicable). See the OpenGL specification for possible values.

Available in Mac OS X v10.3 and later.

Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatOpenGLInternalFormat`
: The OpenGL internal format for this pixel format (if applicable). See the OpenGL specification for possible values.

  Available in Mac OS X v10.3 and later.

  Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatCGBitmapInfo`
: The Core Graphics bitmap information for this pixel format (if applicable).

  Available in Mac OS X v10.3 and later.

  Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatQDCompatibility`
: Indicates whether this format is compatible with QuickDraw (type `CFBoolean`).

  Available in Mac OS X v10.3 and later.

  Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatCGBitmapContextCompatibility`
: Indicates whether this format is compatible with Core Graphics bitmap contexts(type `CFBoolean`).

  Available in Mac OS X v10.3 and later.

  Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatCGImageCompatibility`
: Indicates whether this format is compatible with the `CGImage` type (type `CFBoolean`).

  Available in Mac OS X v10.3 and later.

  Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatOpenGLCompatibility`
: Indicates whether this format is compatible with OpenGL (type `CFBoolean`).

  Available in Mac OS X v10.3 and later.

  Declared in `CVPixelFormatDescription.h`.

`kCVPixelFormatFillExtendedPixelsCallback`
: Specifies a custom extended pixel fill algorithm (type `CFData`). See `CVFillExtendedPixelsCallBack` (page 334) and `CVFillExtendedPixelsCallbackData` (page 337) for more information.

  Available in Mac OS X v10.3 and later.

  Declared in `CVPixelFormatDescription.h`.

Discussion

If you need to define a custom pixel format, you must specify these keys in a Core Foundation dictionary. For information about registering your pixel format, see Technical Q&A 1401: Registering Custom Pixel Formats with QuickTime and Core Video.

In most cases you do not need to specify your own pixel format.

SMPTE State Flags

Flags that describe the SMPTE time state.

```
enum{
    kCVSMPTETimeValid     = (1L << 0),
    kCVSMPTETimeRunning   = (1L << 1)
};
```

Constants

`kCVSMPTETimeValid`

The full time is valid.

Available in Mac OS X v10.3 and later.

Declared in `CVBase.h`.

`kCVSMPTETimeRunning`

Time is running.

Available in Mac OS X v10.3 and later.

Declared in `CVBase.h`.

Discussion

You use these values in the `CVSMPTETime` (page 340) structure.

SMPTE Time Types

Constants that describe the type of SMPTE time.

```
enum{
    kCVSMPTETimeType24          = 0,
    kCVSMPTETimeType25          = 1,
    kCVSMPTETimeType30Drop      = 2,
    kCVSMPTETimeType30          = 3,
    kCVSMPTETimeType2997        = 4,
    kCVSMPTETimeType2997Drop    = 5,
    kCVSMPTETimeType60          = 6,
    kCVSMPTETimeType5994        = 7
};
```

Constants

`kCVSMPTETimeType24`

24 frames per second (standard film).

Available in Mac OS X v10.3 and later.

Declared in `CVBase.h`.

`kCVSMPTETimeType25`

25 frames per second (standard PAL).

Available in Mac OS X v10.3 and later.

Declared in `CVBase.h`.

`kCVSMPTETimeType30Drop`

30 drop frame.

Available in Mac OS X v10.3 and later.

Declared in `CVBase.h`.

`kCVSMPTETimeType30`
  30 frames per second.

  Available in Mac OS X v10.3 and later.

  Declared in `CVBase.h`.

`kCVSMPTETimeType2997`
  29.97 frames per second (standard NTSC).

  Available in Mac OS X v10.3 and later.

  Declared in `CVBase.h`.

`kCVSMPTETimeType2997Drop`
  29.97 drop frame.

  Available in Mac OS X v10.3 and later.

  Declared in `CVBase.h`.

`kCVSMPTETimeType60`
  60 frames per second.

  Available in Mac OS X v10.3 and later.

  Declared in `CVBase.h`.

`kCVSMPTETimeType5994`
  59.94 frames per second.

  Available in Mac OS X v10.3 and later.

  Declared in `CVBase.h`.

Discussion
You use these values in the `CVSMPTETime` (page 340) structure.

Result Codes

The table below lists the result codes returned for Core Video. Note that these result codes are of type `CVReturn`, not type `OSErr`.

Result Code	Value	Description
`kCVReturnSuccess`	0	No error Available in Mac OS X v10.3 and later.
`kCVReturnFirst`	-6660	Placeholder to mark the beginning of Core Video result codes (not returned by any functions). Available in Mac OS X v10.3 and later.
`kCVReturnError`	-6660	An otherwise undefined error occurred. Available in Mac OS X v10.3 and later.
`kCVReturnInvalidArgument`	-6661	Invalid function parameter. For example, out of range or the wrong type. Available in Mac OS X v10.3 and later.

Result Code	Value	Description
`kCVReturnAllocationFailed`	-6662	Memory allocation for a buffer or buffer pool failed. Available in Mac OS X v10.3 and later.
`kCVReturnInvalidDisplay`	-6670	The display specified when creating a display link is invalid. Available in Mac OS X v10.3 and later.
`kCVReturnDisplayLinkAlreadyRunning`	-6671	The specified display link is already running. Available in Mac OS X v10.3 and later.
`kCVReturnDisplayLinkNotRunning`	-6672	The specified display link is not running. Available in Mac OS X v10.3 and later.
`kCVReturnDisplayLinkCallbacksNotSet`	-6673	No callback registered for the specified display link. You must set either the output callback or both the render and display callbacks. Available in Mac OS X v10.3 and later.
`kCVReturnInvalidPixelFormat`	-6680	The buffer does not support the specified pixel format. Available in Mac OS X v10.3 and later.
`kCVReturnInvalidSize`	-6681	The buffer cannot support the requested buffer size (usually too big). Available in Mac OS X v10.3 and later.
`kCVReturnInvalidPixelBufferAttributes`	-6682	A buffer cannot be created with the specified attributes. Available in Mac OS X v10.3 and later.
`kCVReturnPixelBufferNotOpenGLCompatible`	-6683	The pixel buffer is not compatible with OpenGL due to an unsupported buffer size, pixel format, or attribute. Available in Mac OS X v10.3 and later.
`kCVReturnPoolAllocationFailed`	-6690	Allocation for a buffer pool failed, most likely due to a lack of resources. Check to make sure your parameters are in range. Available in Mac OS X v10.3 and later.
`kCVReturnInvalidPoolAttributes`	-6691	A buffer pool cannot be created with the specified attributes. Available in Mac OS X v10.3 and later.

Result Code	Value	Description
`kCVReturnLast`	-6699	Placeholder to mark the end of Core Video result codes (not returned by any functions). Available in Mac OS X v10.3 and later.

Core Animation Function Reference

Framework:	QuartzCore/QuartzCore.h
Declared in	CABase.h CATransform3D.h

Overview

Functions by Task

Timing Functions

`CACurrentMediaTime` (page 362)
Returns the current absolute time, in seconds.

Transform Functions

`CATransform3DIsIdentity` (page 363)
Returns a Boolean value that indicates whether the transform is the identity transform.

`CATransform3DEqualToTransform` (page 362)
Returns a Boolean value that indicates whether the two transforms are exactly equal.

`CATransform3DMakeTranslation` (page 365)
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 364)
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 364)
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 365)
Translate 't' by '(tx, ty, tz)' and return the result: * t' = translate(tx, ty, tz) * t.

`CATransform3DScale` (page 365)
Scale 't' by '(sx, sy, sz)' and return the result: * t' = scale(sx, sy, sz) * t.

`CATransform3DRotate` (page 365)
Rotate 't' by 'angle' radians about the vector '(x, y, z)' and return the result. If the vector has zero length the behavior is undefined: t' = rotation(angle, x, y, z) * t.

`CATransform3DConcat` (page 362)
    Concatenate 'b' to 'a' and return the result: t' = a * b.

`CATransform3DInvert` (page 363)
    Invert 't' and return the result. Returns the original matrix if 't' has no inverse.

`CATransform3DMakeAffineTransform` (page 364)
    Return a transform with the same effect as affine transform 'm'.

`CATransform3DIsAffine` (page 363)
    Returns true if 't' can be exactly represented by an affine transform.

`CATransform3DGetAffineTransform` (page 363)
    Returns the affine transform represented by 't'. If 't' can not be exactly represented as an affine transform the returned value is undefined.

Functions

CACurrentMediaTime

Returns the current absolute time, in seconds.

```
CFTimeInterval CACurrentMediaTime (void);
```

Return Value
A `CFTimeInterval` derived by calling `mach_absolute_time()` and converting the result to seconds.

Availability
Available in 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.

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' = rotation(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' = 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.

Declared In
`CATransform3D.h`

CATransform3DTranslate

Translate 't' by '(tx, ty, tz)' and return the result: * t' = translate(tx, ty, tz) * t.

```
CATransform3D CATransform3DTranslate (CATransform3D t, CGFloat tx, CGFloat ty,
CGFloat tz);
```

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

Declared In
`CATransform3D.h`

Document Revision History

This table describes the changes to *Quartz Core Framework Reference*.

Date	Notes
2008-03-12	Added links to missing classes.
2007-02-17	Added two Core Image documents and the Core Animation classes.
2006-05-23	First publication of this content as a collection of separate documents.

Index