Quartz Core Reference Update

Graphics & Imaging > Quartz

2007-07-18

Contents

Introduction to Quartz Core Reference Update

This document summarizes the symbols that have been added to the Quartz Core framework. The full reference documentation notes in what version a symbol was introduced, but sometimes it's useful to see only the new symbols for a given release.

If you are not familiar with this framework you should refer to the complete framework reference documentation.

Organization of This Document

Symbols are grouped by class or protocol for Objective-C and by header file for C. For each symbol there is a link to complete documentation, if available, and a brief description, if available.

See Also

For reference documentation on this framework, see *Quartz Core Framework Reference*.

10.5 Symbol Changes

This article lists the symbols added to `QuartzCore.framework` in Mac OS X v10.5.

Classes

All of the classes with new symbols are listed alphabetically, with their new class, instance, and delegate methods described.

CAAnimation (New)

Complete reference information is available in the `CAAnimation` reference.

Class Methods

`animation`	Creates and returns a new CAAnimation instance.
`defaultValueForKey:`	Specifies the default value of the property with the specified key.

Instance Methods

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

Delegate Methods

`animationDidStart:`	Called when the animation begins its active duration.
`animationDidStop:finished:`	Called when the animation completes its active duration or is removed from the object it is attached to.

CAConstraint (New)

Complete reference information is available in the `CAConstraint` reference.

Class Methods

`constraintWithAttribute:relativeTo:attribute:`	Creates and returns an CAConstraint object with the specified parameters.
`constraintWithAttribute:relativeTo:attribute:offset:`	Creates and returns an CAConstraint object with the specified parameters.
`constraintWithAttribute:relativeTo:attribute:scale:offset:`	Creates and returns an CAConstraint object with the specified parameters.

Instance Methods

`initWithAttribute:relativeTo:attribute:scale:offset:`	Returns an CAConstraint object with the specified parameters. Designated initializer.

CAConstraintLayoutManager (New)

Complete reference information is available in the `CAConstraintLayoutManager` reference.

Class Methods

`layoutManager`	Creates and returns a new CAConstraintLayoutManager instance.

CALayer (New)

Complete reference information is available in the `CALayer` reference.

Class Methods

`defaultActionForKey:`	Returns an object that implements the default action for the specified identifier.
`defaultValueForKey:`	Specifies the default value of the property with the specified key.
`layer`	Creates and returns an instance of CALayer.

Instance Methods

`actionForKey:`	Returns an object that implements the action for the specified identifier.
`addAnimation:forKey:`	Add an animation object to the receiver's render tree for the specified key.

`addConstraint:`	Adds the constraint to the receiver's array of constraint objects.
`addSublayer:`	Appends the layer to the receiver's sublayers array.
`affineTransform`	Convenience method for getting the transform property as an affine transform.
`animationForKey:`	Returns the animation added to the receiver with the specified identifier.
`containsPoint:`	Returns whether the receiver contains a specified point.
`convertPoint:fromLayer:`	Converts the point from the specified layer's coordinate system to the receiver's coordinate system.
`convertPoint:toLayer:`	Converts the point from the receiver's coordinate system to the specified layer's coordinate system.
`convertRect:fromLayer:`	Converts the rectangle from the specified layer's coordinate system to the receiver's coordinate system.
`convertRect:toLayer:`	Converts the rectangle from the receiver's coordinate system to the specified layer's coordinate system.
`convertTime:fromLayer:`	Converts the time interval from the specified layer's time space to the receiver's time space.
`convertTime:toLayer:`	Converts the time interval from the receiver's time space to the specified layer's time space
`display`	Reload the content of this layer.
`drawInContext:`	Draws the receiver's content in the specified graphics context.
`hitTest:`	Returns the farthest descendant of the receiver in the layer hierarchy (including itself) that contains a specified point.
`init`	
`initWithLayer:`	Override to copy or initialize custom fields of the specified layer.
`insertSublayer:above:`	Inserts the layer into the receiver's sublayers array, above the specified sublayer.
`insertSublayer:atIndex:`	Inserts the layer as a sublayer of the receiver at the specified index.
`insertSublayer:below:`	Inserts the layer into the receiver's sublayers array, below the specified sublayer.
`layoutIfNeeded`	Recalculate the receiver's layout, if required.
`layoutSublayers`	Called when the layer requires layout.
`modelLayer`	Returns the model layer of the receiver, if it represents a current presentation layer.

`preferredFrameSize`	Returns the preferred frame size of the layer in the coordinate space of the superlayer.
`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.
`removeAllAnimations`	Remove all animations attached to the receiver.
`removeAnimationForKey:`	Remove the animation attached to the receiver with the specified key.
`removeFromSuperlayer`	Removes the layer from the sublayers array or mask property of the receiver's superlayer.
`renderInContext:`	Renders the receiver and its sublayers into the specified context.
`replaceSublayer:with:`	Replaces the layer in the receiver's sublayers array with the specified new layer.
`resizeSublayersWithOldSize:`	Informs the receiver's sublayers that the receiver's bounds rectangle size has changed.
`resizeWithOldSuperlayerSize:`	Informs the receiver that the bounds size of its superview has changed.
`scrollPoint:`	Scrolls the receiver's closest ancestor CAScrollLayer so that the specified point lies at the origin of the layer.
`scrollRectToVisible:`	Scrolls the receiver's closest ancestor CAScrollLayer the minimum distance needed so that the specified rectangle becomes visible.
`setAffineTransform:`	Convenience method for setting the transform property as an affine transform.
`setNeedsDisplay`	Marks the receiver as needing display before the content is next committed.
`setNeedsDisplayInRect:`	Marks the region of the receiver within the specified rectangle as needing display.
`setNeedsLayout`	Called when the preferred size of the receiver may have changed.
`shouldArchiveValueForKey:`	Specifies whether the value of the property for a given key is archived.

Delegate Methods

`actionForLayer:forKey:`	Allows the delegate to customize the action for a layer.
`displayLayer:`	Allows the delegate to override the display implementation.

`drawLayer:inContext:`	Allows the delegate to override the layer's drawInContext: implementation.
`invalidateLayoutOfLayer:`	Invalidates the layout of the specified layer.
`layoutSublayersOfLayer:`	Layout each of the sublayers in the specified layer.
`preferredSizeOfLayer:`	Returns the preferred size of the specified layer in its coordinate system.

CAMediaTimingFunction (New)

Complete reference information is available in the `CAMediaTimingFunction` reference.

Class Methods

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

Instance Methods

`getControlPointAtIndex:values:`	Returns the control point for the specified index.

CAOpenGLLayer (New)

Complete reference information is available in the `CAOpenGLLayer` reference.

Instance Methods

`canDrawInCGLContext:pixelFormat: forLayerTime:displayTime:`	Returns whether the receiver should draw OpenGL content for the specified time.
`copyCGLContextForPixelFormat:`	Returns the rendering context the receiver requires for the specified pixel format.
`copyCGLPixelFormatForDisplayMask:`	Returns the OpenGL pixel format suitable for rendering to the set of displays specified by the display mask.
`drawInCGLContext:pixelFormat: forLayerTime:displayTime:`	Draws the OpenGL content for the specified time.
`releaseCGLContext:`	Releases the specified rendering context.
`releaseCGLPixelFormat:`	Releases the specified OpenGL pixel format object.

CAPropertyAnimation (New)

Complete reference information is available in the `CAPropertyAnimation` reference.

Class Methods

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

CARenderer (New)

Complete reference information is available in the `CARenderer` reference.

Class Methods

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

Instance Methods

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

CAScrollLayer (New)

Complete reference information is available in the `CAScrollLayer` reference.

Instance Methods

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

CATiledLayer (New)

Complete reference information is available in the `CATiledLayer` reference.

Class Methods

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

CATransaction (New)

Complete reference information is available in the `CATransaction` reference.

Class Methods

`begin`	Begin a new transaction for the current thread.
`commit`	Commit all changes made during the current transaction.
`flush`	Flushes any extant implicit transaction.
`setValue:forKey:`	Sets the arbitrary keyed-data for the specified key.
`valueForKey:`	Returns the arbitrary keyed-data specified by the given key.

CIContext

Complete reference information is available in the `CIContext` reference.

Instance Methods

`createCGImage:fromRect:format:colorSpace:`	Creates a Quartz 2D image from a region of a CIImage object.
`render:toBitmap:rowBytes:bounds:format:colorSpace:`	Renders to the given bitmap.

CIFilter

Complete reference information is available in the `CIFilter` reference.

Class Methods

`filterWithImageData:options:`	Returns a CIFilter object initialized with RAW image data supplied to the method.
`filterWithImageURL:options:`	Returns a CIFilter object initialized with data from a RAW image file.
`localizedDescriptionForFilterName:`	Returns the localized description of a filter for display in the user interface.
`localizedReferenceDocumentationForFilterName:`	Returns the location of the localized reference documentation that describes the filter.

CIFilterGenerator (New)

Complete reference information is available in the `CIFilterGenerator` reference.

Class Methods

`filterGenerator`	Creates and returns an empty filter generator object.
`filterGeneratorWithContentsOfURL:`	Creates and returns a filter generator object and initializes it with the contents of a filter generator file.

Instance Methods

`classAttributes`	Retrieves the class attributes associated with a filter.
`connectObject:withKey:toObject:withKey:`	Adds an object to the filter chain.
`disconnectObject:withKey:toObject:withKey:`	Removes the connection between two objects in the filter chain.
`exportedKeys`	Returns an array of the exported keys.
`exportKey:fromObject:withName:`	Exports an input or output key of an object in the filter chain.
`filter`	Creates a filter object based on the filter chain.
`initWithContentsOfURL:`	Initializes a filter generator object with the contents of a filter generator file.
`registerFilterName:`	Registers the name associated with a filter chain.
`removeExportedKey:`	Removes a key that was previously exported.

`setAttributes:forExportedKey:`	Sets a dictionary of attributes for an exported key.
`setClassAttributes:`	Seta the class attributes for a filter.
`writeToURL:atomically:`	Archives a filter generator object to a filter generator file.

CIImage

Complete reference information is available in the `CIImage` reference.

Class Methods

`emptyImage`	Creates and returns an empty image object.
`imageWithColor:`	Creates and returns an image of infinite extent that is initialized the specified color.

Instance Methods

`imageByCroppingToRect:`	Returns a new image that represents the original image after cropping to a rectangle.
`initWithColor:`	Initializes an image with the specified color.

CIImageAccumulator

Complete reference information is available in the `CIImageAccumulator` reference.

Instance Methods

`clear`	Resets the accumulator, discarding any pending updates and the current content.

CIVector

Complete reference information is available in the `CIVector` reference.

Instance Methods

`initWithString:`	Initializes a vector with values provided in a string representation.

NSObject

Complete reference information is available in the `NSObject` reference.

Instance Methods

`actionForLayer:forKey:`	Allows the delegate to customize the action for a layer.
`animationDidStart:`	Called when the animation begins its active duration.
`animationDidStop:finished:`	Called when the animation completes its active duration or is removed from the object it is attached to.
`displayLayer:`	Allows the delegate to override the display implementation.
`drawLayer:inContext:`	Allows the delegate to override the layer's drawInContext: implementation.
`invalidateLayoutOfLayer:`	Invalidates the layout of the specified layer.
`layoutSublayersOfLayer:`	Layout each of the sublayers in the specified layer.
`preferredSizeOfLayer:`	Returns the preferred size of the specified layer in its coordinate system.

NSValue

Complete reference information is available in the `NSValue` reference.

Class Methods

`valueWithCATransform3D:`	Creates and returns an NSValue object that contains a given CATransform3D structure.

Instance Methods

`CATransform3DValue`	Returns an CATransform3D structure representation of the receiver.

Protocols

All of the protocols with new symbols are listed alphabetically, with their new methods described.

CAAction (New)

Complete reference information is available in the `CAAction` reference.

Instance Methods

`runActionForKey:object:arguments:`	Called to trigger the action specified by the identifier.

CAMediaTiming (New)

Complete reference information is available in the `CAMediaTiming` reference.

C Symbols

All of the header files with new symbols are listed alphabetically, with their new symbols described.

CAAnimation.h

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`calculationMode`	
`endProgress`	
`kCAAnimationDiscrete`	Each keyframe value is used in turn, no interpolated values are calculated.
`kCAAnimationLinear`	Simple linear calculation between keyframe values.
`kCAAnimationPaced`	Keyframe values are interpolated to produce an even pace throughout the animation. This mode is not currently implemented
`kCAAnimationRotateAuto`	The objects travel on a tangent to the path.
`kCAAnimationRotateAutoReverse`	The objects travel at a 180 degree tangent to the path.
`kCATransitionFade`	The layer's content fades as it becomes visible or hidden.
`kCATransitionFromBottom`	The transition begins at the bottom of the layer.
`kCATransitionFromLeft`	The transition begins at the left side of the layer.

kCATransitionFromRight	The transition begins at the right side of the layer.
kCATransitionFromTop	The transition begins at the top of the layer.
kCATransitionMoveIn	The layer's content slides into place over any existing content. The "Common Transition Subtypes" are used with this transition.
kCATransitionPush	The layer's content pushes any existing content as it slides into place. The "Common Transition Subtypes" are used with this transition.
kCATransitionReveal	The layer's content is revealed gradually in the direction specified by the transition subtype. The "Common Transition Subtypes" are used with this transition.
keyTimes	
rotationMode	
startProgress	
subtype	
timingFunctions	

CABase.h

Functions

All of the new functions in this header file are listed alphabetically, with links to documentation and abstracts, if available.

CACurrentMediaTime	Returns the current absolute time, in seconds.

CAConstraintLayoutManager.h

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

CAConstraintAttribute	The constraint attribute type.
kCAConstraintHeight	The height of a layer.
kCAConstraintMaxX	The right edge of a layer's frame.
kCAConstraintMaxY	The top edge of a layer's frame.

kCAConstraintMidX	The horizontal location of the center of a layer's frame.
kCAConstraintMidY	The vertical location of the center of a layer's frame.
kCAConstraintMinX	The left edge of a layer's frame.
kCAConstraintMinY	The bottom edge of a layer's frame.
kCAConstraintWidth	The width of a layer.
_CAConstraintAttribute	These constants represent the geometric edge or axis of a constraint.

CALayer.h

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

CAAutoresizingMask	These constants are used by the autoresizingMask property.
CAEdgeAntialiasingMask	This mask is used by the edgeAntialiasingMask property.
kCAFilterLinear	Linear interpolation filter.
kCAFilterNearest	Nearest neighbor interpolation filter.
kCAGravityBottom	The content is horizontally centered at the bottom-edge of the bounds rectangle.
kCAGravityBottomLeft	The content is positioned in the bottom-left corner of the bounds rectangle.
kCAGravityBottomRight	The content is positioned in the bottom-right corner of the bounds rectangle.
kCAGravityCenter	The content is horizontally and verticallycentered in the bounds rectangle.
kCAGravityLeft	The content is vertically centered at the left-edge of the bounds rectangle.
kCAGravityResize	The content is resized to fit the entire bounds rectangle.
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.
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.

`kCAGravityRight`	The content is vertically centered at the right-edge of the bounds rectangle.
`kCAGravityTop`	The content is horizontally centered at the top-edge of the bounds rectangle.
`kCAGravityTopLeft`	The content is positioned in the top-left corner of the bounds rectangle.
`kCAGravityTopRight`	The content is positioned in the top-right corner of the bounds rectangle.
`kCALayerBottomEdge`	Specifies that the bottom edge of the receiver's content should be antialiased.
`kCALayerHeightSizable`	The receiver's height is flexible.
`kCALayerLeftEdge`	Specifies that the left edge of the receiver's content should be antialiased.
`kCALayerMaxXMargin`	The right margin between the receiver and its superview is flexible.
`kCALayerMaxYMargin`	The top margin between the receiver and its superview is flexible.
`kCALayerMinXMargin`	The left margin between the receiver and its superview is flexible.
`kCALayerMinYMargin`	The bottom margin between the receiver and its superview is flexible.
`kCALayerNotSizable`	The receiver cannot be resized.
`kCALayerRightEdge`	Specifies that the right edge of the receiver's content should be antialiased.
`kCALayerTopEdge`	Specifies that the top edge of the receiver's content should be antialiased.
`kCALayerWidthSizable`	The receiver's width is flexible.
`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.
`kCAOnOrderOut`	The identifier that represents the action taken when the layer is removed from the layer hierarchy or is hidden.
`kCATransition`	The identifier that represents a transition animation.

CAMediaTiming.h

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`kCAFillModeBackwards`	The receiver clamps values before zero to zero when the animation is completed.
`kCAFillModeBoth`	The receiver clamps values at both ends of the object's time space
`kCAFillModeForwards`	The receiver remains visible in its final state when the animation is completed.
`kCAFillModeRemoved`	The receiver is removed from the presentation when the animation is completed.

CAMediaTimingFunction.h

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`kCAMediaTimingFunctionEaseIn`	Specifies ease-in pacing. Ease-in pacing causes the animation to begin slowly, and then speed up as it progresses.
`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.
`kCAMediaTimingFunctionEaseOut`	Specifies ease-out pacing. An ease-out pacing causes the animation to begin quickly, and then slow as it completes.
`kCAMediaTimingFunctionLinear`	Specifies linear pacing. A linear pacing causes an animation to occur evenly over its duration.

CAScrollLayer.h

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`kCAScrollBoth`	The receiver is able to scroll both horizontally and vertically.
`kCAScrollHorizontally`	The receiver is able to scroll horizontally.
`kCAScrollNone`	The receiver is unable to scroll.
`kCAScrollVertically`	The receiver is able to scroll vertically.

CATextLayer.h

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`kCAAlignmentCenter`	Text is visually center aligned.
`kCAAlignmentJustified`	Text is justified.
`kCAAlignmentLeft`	Text is visually left aligned.
`kCAAlignmentNatural`	Use the natural alignment of the text's script.
`kCAAlignmentRight`	Text is visually right aligned.
`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.
`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.
`kCATruncationNone`	If the wrapped property is YES, the text is wrapped to the receiver's bounds, otherwise the text is clipped to the receiver's bounds.
`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.

CATransaction.h

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`kCATransactionAnimationDuration`	Default duration, in seconds, for animations added to layers. The value for this key must be an instance of NSNumber.

`kCATransactionDisableActions`	If YES, implicit actions for property changes are suppressed. The value for this key must be an instance of NSNumber.

CATransform3D.h

Functions

All of the new functions in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`CATransform3DConcat`	Concatenate 'b' to 'a' and return the result: t' = a * b.
`CATransform3DEqualToTransform`	Returns a Boolean value that indicates whether the two transforms are exactly equal.
`CATransform3DGetAffineTransform`	Returns the affine transform represented by 't'. If 't' can not be exactly represented as an affine transform the returned value is undefined.
`CATransform3DInvert`	Invert 't' and return the result. Returns the original matrix if 't' has no inverse.
`CATransform3DIsAffine`	Returns true if 't' can be exactly represented by an affine transform.
`CATransform3DIsIdentity`	Returns a Boolean value that indicates whether the transform is the identity transform.
`CATransform3DMakeAffineTransform`	Return a transform with the same effect as affine transform 'm'.
`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.
`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].
`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].
`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.
`CATransform3DScale`	Scale 't' by '(sx, sy, sz)' and return the result: * t' = scale(sx, sy, sz) * t.
`CATransform3DTranslate`	Translate 't' by '(tx, ty, tz)' and return the result: * t' = translate(tx, ty, tz) * t.

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`CATransform3D`	Defines the standard transform matrix used throughout Core Animation.
`CATransform3DIdentity`	The identity transform: [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1].
`m13`	
`m14`	
`m23`	
`m24`	
`m31`	
`m32`	
`m33`	
`m34`	
`m41`	
`m42`	
`m43`	
`m44`	

CIFilter.h

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`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.
`kCIAttributeReferenceDocumentation`	The localized reference documentation for the filter. The reference should provide developers with technical details.
`kCIAttributeTypeCount`	A positive integer value.
`kCIAttributeTypeInteger`	An integer value.

`kCICategoryFilterGenerator`	A filter created by chaining several filters together and then packaged as a CIFilterGenerator object.
`kCICategoryReduction`	A filter that reduces image data. These filters are used to solve image analysis problems.
`kCIInputAngleKey`	A key for a scalar value (NSNumber) that specifies an angle.
`kCIInputAspectRatioKey`	A key for a scalar value (NSNumber) that specifies a ratio.
`kCIInputBackgroundImageKey`	A key for the CIImage object to use as a background image.
`kCIInputBrightnessKey`	A key for a scalar value (NSNumber) that specifies a brightness level.
`kCIInputCenterKey`	A key for a CIVector object that specifies the center of the area, as x and y- coordinates, to be filtered.
`kCIInputColorKey`	A key for a CIColor object that specifies a color value.
`kCIInputContrastKey`	A key for a scalar value (NSNumber) that specifies a contrast level.
`kCIInputEVKey`	A key for a scalar value (NSNumber) that specifies how many F-stops brighter or darker the image should be.
`kCIInputExtentKey`	A key for a CIVector object that specifies a rectangle that defines the extent of the effect.
`kCIInputGradientImageKey`	A key for a CIImage object that specifies an environment map with alpha. Typically, this image contains highlight and shadow.
`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.
`kCIInputIntensityKey`	A key for a scalar value (NSNumber) that specifies an intensity value.
`kCIInputMaskImageKey`	A key for a CIImage object to use as a mask.
`kCIInputRadiusKey`	A key for a scalar value (NSNumber) that specifies that specifies the distance from the center of an effect.
`kCIInputRefractionKey`	A key for a scalar value (NSNumber) that specifies the index of refraction of the material (such as glass) used in the effect.
`kCIInputSaturationKey`	A key for a scalar value (NSNumber) that specifies the amount to adjust the saturation.
`kCIInputScaleKey`	A key for a scalar value (NSNumber) that specifies the amount of the effect.

`kCIInputShadingImageKey`	A key for a CIImage object that specifies an environment map with alpha values. Typically this image contains highlight and shadow.
`kCIInputSharpnessKey`	A key for a scalar value (NSNumber) that specifies the amount of sharpening to apply.
`kCIInputTargetImageKey`	A key for a CIImage object that is the target image for a transition.
`kCIInputTimeKey`	A key for z scalar value (NSNumber) that specifies a time.
`kCIInputTransformKey`	A key for an NSAffineTransform object that specifies a transformation to apply.
`kCIInputWidthKey`	A key for a scalar value (NSNumber) that specifies the width of the effect.
`kCIOutputImageKey`	A key for the CIImage object produced by a filter.
`kCIUIParameterSet`	The set of input parameters to use. The associated value can be kCIUISetBasic, kCIUISetIntermediate, kCIUISetAdvanced, or kCIUISetDevelopment.
`kCIUISetAdvanced`	Controls that are appropriate for an advanced user scenario.
`kCIUISetBasic`	Controls that are appropriate for a basic user scenario, that is, the minimum of settings to control the filter.
`kCIUISetDevelopment`	Controls that should be visible only for development purposes.
`kCIUISetIntermediate`	Controls that are appropriate for an intermediate user scenario.

CIFilterGenerator.h

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

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

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

CIRAWFilter.h

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`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.
`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.
`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.
`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.

`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.
`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.
`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.
`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.
`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.

`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.
`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.
`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).
`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.
`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.
`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.
`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..

10.4 Symbol Changes

This article lists the symbols added to `QuartzCore.framework` in Mac OS X v10.4.

Classes

All of the classes with new symbols are listed alphabetically, with their new class, instance, and delegate methods described.

CIColor (New)

Complete reference information is available in the `CIColor` reference.

Class Methods

`colorWithCGColor:`	Creates a color object from a Quartz color.
`colorWithRed:green:blue:`	Creates a color object using the specified RGB color component values
`colorWithRed:green:blue:alpha:`	Creates a color object using the specified RGBA color component values.
`colorWithString:`	Creates a color object using the RGBA color component values specified by a string.

Instance Methods

`alpha`	Returns the alpha value of the color.
`blue`	Returns the blue component of the color.
`colorSpace`	Returns the Quartz 2D color space associated with the color.
`components`	Returns the color components of the color.
`green`	Returns the green component of the color.
`initWithCGColor:`	Initializes a color object with a Quartz color.
`numberOfComponents`	Returns the number of color components in the color.

`red`	Returns the red component of the color.
`stringRepresentation`	Returns a formatted string that specifies the components of the color.

CIContext (New)

Complete reference information is available in the `CIContext` reference.

Class Methods

`contextWithCGContext:options:`	Creates a Core Image context from a Quartz context, using the specified options.
`contextWithCGLContext:pixelFormat:options:`	Creates a Core Image context from a CGL context, using the specified options and pixel format object.

Instance Methods

`clearCaches`	Frees any cached data, such as temporary images, associated with the context and runs the garbage collector.
`createCGImage:fromRect:`	Creates a Quartz 2D image from a region of a CIImage object.
`createCGLayerWithSize:info:`	Creates a CGLayer object from the provided parameters.
`drawImage:atPoint:fromRect:`	Renders a region of an image to a point in the context destination.
`drawImage:inRect:fromRect:`	Renders a region of an image to a rectangle in the context destination.
`reclaimResources`	Runs the garbage collector to reclaim any resources that the context no longer requires.

CIFilter (New)

Complete reference information is available in the `CIFilter` reference.

Class Methods

`filterNamesInCategories:`	Returns an array of all published filter names that match all the specified categories.
`filterNamesInCategory:`	Returns an array of all published filter names in the specified category.

`filterWithName:`	Creates a CIFilter object for a specific kind of filter.
`filterWithName:keysAndValues:`	Creates a CIFilter object for a specific kind of filter and initializes the input values.
`localizedNameForCategory:`	Returns the localized name for the specified filter category.
`localizedNameForFilterName:`	Returns the localized name for the specified filter name.
`registerFilterName:constructor:classAttributes:`	Publishes a custom filter that is not packaged as an image unit.

Instance Methods

`apply:`	Produces a CIImage object by applying a kernel function.
`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.
`attributes`	Returns a dictionary of key-value pairs that describe the filter.
`inputKeys`	Returns an array that contains the names of the input parameters to the filter.
`outputKeys`	Returns an array that contains the names of the output parameters for the filter.
`setDefaults`	Sets all input values for a filter to default values.

CIFilterShape (New)

Complete reference information is available in the `CIFilterShape` reference.

Class Methods

`shapeWithRect:`	Creates a filter shape object and initializes it with a rectangle.

Instance Methods

`initWithRect:`	Initializes a filter shape object with a rectangle.
`insetByX:Y:`	Modifies a filter shape object so that it is inset by the specified x and y values.

`intersectWith:`	Creates a filter shape object that represents the intersection of the current filter shape and the specified filter shape object.
`intersectWithRect:`	Creates a filter shape that represents the intersection of the current filter shape and a rectangle.
`transformBy:interior:`	Creates a filter shape that results from applying a transform to the current filter shape.
`unionWith:`	Creates a filter shape that results from the union of the current filter shape and another filter shape object.
`unionWithRect:`	Creates a filter shape that results from the union of the current filter shape and a rectangle.

CIImage (New)

Complete reference information is available in the `CIImage` reference.

Class Methods

`imageWithBitmapData:bytesPerRow:size:format:colorSpace:`	Creates and returns an image object from bitmap data.
`imageWithCGImage:`	Creates and returns an image object from a Quartz 2D image.
`imageWithCGImage:options:`	Creates and returns an image object from a Quartz 2D image using the specified color space.
`imageWithCGLayer:`	Creates and returns an image object from the contents supplied by a CGLayer object.
`imageWithCGLayer:options:`	Creates and returns an image object from the contents supplied by a CGLayer object, using the specified options.
`imageWithContentsOfURL:`	Creates and returns an image object from the contents of a file.
`imageWithContentsOfURL:options:`	Creates and returns an image object from the contents of a file, using the specified options.
`imageWithCVImageBuffer:`	Creates and returns an image object from the contents of CVImageBuffer object.
`imageWithCVImageBuffer:options:`	Creates and returns an image object from the contents of CVImageBuffer object, using the specified options.

`imageWithData:`	Creates and returns an image object initialized with the supplied image data.
`imageWithData:options:`	Creates and returns an image object initialized with the supplied image data, using the specified options.
`imageWithImageProvider:size:width:format:colorSpace:options:`	
`imageWithTexture:size:flipped:colorSpace:`	Creates and returns an image object initialized with data supplied by an OpenGL texture.

Instance Methods

`definition`	Returns a filter shape object that represents the domain of definition of the image.
`extent`	Returns a rectangle that specifies the extent of the image.
`imageByApplyingTransform:`	Returns a new image that represents the original image after applying an affine transform.
`initWithBitmapData:bytesPerRow:size:format:colorSpace:`	Initializes an image object with bitmap data.
`initWithCGImage:`	Initializes an image object with a Quartz 2D image.
`initWithCGImage:options:`	Initializes an image object with a Quartz 2D image, using the specified options.
`initWithCGLayer:`	Initializes an image object from the contents supplied by a CGLayer object.
`initWithCGLayer:options:`	Initializes an image object from the contents supplied by a CGLayer object, using the specified options.
`initWithContentsOfURL:`	Initializes an image object from the contents of a file.
`initWithContentsOfURL:options:`	Initializes an image object from the contents of a file, using the specified options.
`initWithCVImageBuffer:`	Initializes an image object from the contents of CVImageBuffer object.
`initWithCVImageBuffer:options:`	Initializes an image object from the contents of CVImageBuffer object, using the specified options.
`initWithData:`	Initializes an image object with the supplied image data.

`initWithData:options:`	Initializes an image object with the supplied image data, using the specified options.
`initWithImageProvider:size:width:format:colorSpace:options:`	
`initWithTexture:size:flipped:colorSpace:`	Initializes an image object with data supplied by an OpenGL texture.

CIImageAccumulator (New)

Complete reference information is available in the `CIImageAccumulator` reference.

Class Methods

`imageAccumulatorWithExtent:format:`	Creates an image accumulator with the specified extent and pixel format.

Instance Methods

`extent`	Returns the extent of the image associated with the image accumulator.
`format`	Returns the pixel format of the image accumulator.
`image`	Returns the current contents of the image accumulator.
`initWithExtent:format:`	Initializes an image accumulator with the specified extent and pixel format.
`setImage:`	Sets the contents of the image accumulator to the contents of the specified image object.
`setImage:dirtyRect:`	Updates an image accumulator with a subregion of an image object.

CIKernel (New)

Complete reference information is available in the `CIKernel` reference.

Class Methods

`kernelsWithString:`	Creates and returns and array of CIKernel objects.

Instance Methods

`name`	Returns the name of a kernel routine.

`setROISelector:`	Sets the selector used to query the region of interest of the kernel.

CIPlugIn (New)

Complete reference information is available in the `CIPlugIn` reference.

Class Methods

`loadAllPlugIns`	Scans directories for files that have the .plugin extension and then loads the image units.
`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.
`loadPlugIn:allowNonExecutable:`	Loads filters from an image unit that have the appropriate executable status.

CISampler (New)

Complete reference information is available in the `CISampler` reference.

Class Methods

`samplerWithImage:`	Creates and returns a sampler that references an image.
`samplerWithImage:keysAndValues:`	Creates and returns a sampler that references an image using options specified as key-value pairs.
`samplerWithImage:options:`	Creates and returns a sampler that references an image using options specified in a dictionary.

Instance Methods

`definition`	Gets the domain of definition (DOD) of the sampler.
`extent`	Gets the rectangle that specifies the extent of the sampler.
`initWithImage:`	Initializes a sampler with an image object.
`initWithImage:keysAndValues:`	Initializes the sampler with an image object using options specified as key-value pairs.
`initWithImage:options:`	Initializes the sampler with an image object using options specified in a dictionary.

CIVector (New)

Complete reference information is available in the `CIVector` reference.

Class Methods

`vectorWithString:`	Creates and returns a vector that is initialized with values provided in a string representation.
`vectorWithValues:count:`	Creates and returns a vector that is initialized with the specified values.
`vectorWithX:`	Creates and returns a vector that is initialized with one value.
`vectorWithX:Y:`	Creates and returns a vector that is initialized with two values.
`vectorWithX:Y:Z:`	Creates and returns a vector that is initialized with three values.
`vectorWithX:Y:Z:W:`	Creates and returns a vector that is initialized with four values.

Instance Methods

`count`	Returns the number of items in a vector.
`initWithValues:count:`	Initializes a vector with the provided values.
`initWithX:`	Initializes the first position of a vector with the provided values.
`initWithX:Y:`	Initializes the first two positions of a vector with the provided values.
`initWithX:Y:Z:`	Initializes the first three positions of a vector with the provided values.
`initWithX:Y:Z:W:`	Initializes four positions of a vector with the provided values.
`stringRepresentation`	Returns a string representation for a vector.
`valueAtIndex:`	Returns a value from a specific position in a vector.
`W`	Returns the value located in the fourth position in a vector.
`X`	Returns the value located in the first position in a vector.
`Y`	Returns the value located in the second position in a vector.
`Z`	Returns the value located in the third position in a vector.

NSObject

Complete reference information is available in the `NSObject` reference.

Instance Methods

`provideImageData:bytesPerRow:origin: x:size:width:userInfo:`	

Protocols

All of the protocols with new symbols are listed alphabetically, with their new methods described.

CIPlugInRegistration (New)

Complete reference information is available in the `CIPlugInRegistration` reference.

Instance Methods

`load:`	Loads and initializes an image unit, performing custom tasks as needed.

C Symbols

All of the header files with new symbols are listed alphabetically, with their new symbols described.

CIFilter.h

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

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

`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)).
`kCIAttributeDefault`	The default value, specified as a floating-point value, for a filter parameter.
`kCIAttributeDisplayName`	The localized display name of the attribute.
`kCIAttributeFilterCategories`	An array of filter category keys that specifies all the categories in which the filter is a member.
`kCIAttributeFilterDisplayName`	The localized version of the filter name that is displayed in the user interface.
`kCIAttributeFilterName`	The filter name, specified as an NSString object.
`kCIAttributeIdentity`	If supplied as a value for a parameter, the parameter has no effect on the input image.
`kCIAttributeMax`	The maximum value for a filter parameter, specified as a floating-point value.
`kCIAttributeMin`	The minimum value for a filter parameter, specified as a floating-point value.
`kCIAttributeName`	The name of the attribute.
`kCIAttributeSliderMax`	The maximum value, specified as a floating-point value, to use for a slider that controls input values for a filter parameter.
`kCIAttributeSliderMin`	The minimum value, specified as a floating-point value, to use for a slider that controls input values for a filter parameter.
`kCIAttributeType`	The attribute type.
`kCIAttributeTypeAngle`	An angle.
`kCIAttributeTypeBoolean`	A Boolean value.
`kCIAttributeTypeDistance`	A distance.
`kCIAttributeTypeGradient`	An n-by-1 gradient image used to describe a color ramp.
`kCIAttributeTypeOffset`	An offset. (A 2-element vector type.)
`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.

`kCIAttributeTypePosition`	A two-dimensional location in the working coordinate space. (A 2-element vector type.)
`kCIAttributeTypePosition3`	A three-dimensional location in the working coordinate space. (A 3-element vector type.)
`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.)
`kCIAttributeTypeScalar`	A scalar value.
`kCIAttributeTypeTime`	A parametric time for transitions, specified as a floating-point value in the range of 0.0 to 1.0.
`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.
`kCICategoryBuiltIn`	A filter provided by Core Image. This distinguishes built-in filters from plug-in filters.
`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.
`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.
`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.
`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.
`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.

`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.
`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.
`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.
`kCICategoryHighDynamicRange`	A filter that works on high dynamic range pixels.
`kCICategoryInterlaced`	A filter that works on interlaced images.
`kCICategoryNonSquarePixels`	A filter that works on non-square pixels.
`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.
`kCICategoryStillImage`	A filter that works on still images.
`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.
`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.
`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.
`kCICategoryVideo`	A filter that works on video images.

CIImageProvider.h

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

kCIImageProviderTileSize	A key for the image tiles size. The associated value is an NSArray that containsNSNumber objects for the dimensions of the image tiles requested from the image provider.
kCIImageProviderUserInfo	A key for data needed by the image provider. The associated value is an object that contains the needed data.

CISampler.h

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

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.
kCISamplerFilterLinear	Bilinear interpolation.
kCISamplerFilterMode	The key for the filtering to use when sampling the image. Possible values are kCISamplerFilterNearest and kCISamplerFilterLinear.
kCISamplerFilterNearest	Nearest neighbor sampling.
kCISamplerWrapBlack	Pixels are transparent black.
kCISamplerWrapClamp	Coordinates are clamped to the extent.
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 and kCISamplerWrapClamp.

10.3 Symbol Changes

This article lists the symbols added to `QuartzCore.framework` in Mac OS X v10.3.

Classes

All of the classes with new symbols are listed alphabetically, with their new class, instance, and delegate methods described.

C Symbols

All of the header files with new symbols are listed alphabetically, with their new symbols described.

CVBase.h

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`CVOptionFlags`	Define flags to be used for the display link output callback function.
`CVSMPTETime`	A structure for holding a SMPTE time.
`hostTime`	
`kCVIndefiniteTime`	An unknown or indefinite time. For example, CVDisplayLinkGetNominalOutputVideoRefreshPeriod returns kCVIndefiniteTime if the display link specified is not valid.
`kCVSMPTETimeRunning`	Time is running.
`kCVSMPTETimeType24`	24 frames per second (standard film).
`kCVSMPTETimeType25`	25 frames per second (standard PAL).
`kCVSMPTETimeType2997`	29.97 frames per second (standard NTSC).

`kCVSMPTETimeType2997Drop`	29.97 drop frame.
`kCVSMPTETimeType30`	30 frames per second.
`kCVSMPTETimeType30Drop`	30 drop frame.
`kCVSMPTETimeType5994`	59.94 frames per second.
`kCVSMPTETimeType60`	60 frames per second.
`kCVSMPTETimeValid`	The full time is valid.
`kCVTimeIsIndefinite`	The time value is unknown.
`kCVTimeStampBottomField`	The timestamp represents the bottom lines of an interlaced image.
`kCVTimeStampHostTimeValid`	The value in the host time field is valid.
`kCVTimeStampIsInterlaced`	A convenience constant indicating that the timestamp is for an interlaced image.
`kCVTimeStampRateScalarValid`	The value in the rate scalar field is valid.
`kCVTimeStampSMPTETimeValid`	The value in the SMPTE time field is valid.
`kCVTimeStampTopField`	The timestamp represents the top lines of an interlaced image.
`kCVTimeStampVideoHostTimeValid`	A convenience constant indicating that both the video time and host time fields are valid.
`kCVTimeStampVideoRefreshPeriodValid`	The value in the video refresh period field is valid.
`kCVTimeStampVideoTimeValid`	The value in the video time field is valid.
`kCVZeroTime`	Zero time or duration. For example, CVDisplayLinkGetOutputVideoLatency returns kCVZeroTime for zero video latency.
`rateScalar`	
`smpteTime`	
`subframeDivisor`	
`subframes`	
`videoRefreshPeriod`	
`videoTime`	
`videoTimeScale`	

CVBuffer.h

Functions

All of the new functions in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`CVBufferGetAttachment`	Returns a specific attachment of a Core Video buffer.
`CVBufferGetAttachments`	Returns all attachments of a Core Video buffer.
`CVBufferPropagateAttachments`	Copies all propagatable attachments from one Core Video buffer to another.
`CVBufferRelease`	Releases a Core Video buffer.
`CVBufferRemoveAllAttachments`	Removes all attachments of a Core Video buffer.
`CVBufferRemoveAttachment`	Removes a specific attachment of a Core Video buffer.
`CVBufferRetain`	Retains a Core Video buffer.
`CVBufferSetAttachment`	Sets or adds an attachment of a Core Video buffer.
`CVBufferSetAttachments`	Sets a set of attachments for a Core Video buffer.

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`CVAttachmentMode`	Specify the propagation mode of a Core Video buffer attachment.
`CVBufferRef`	Defines the base type for all Core Video buffers.
`kCVAttachmentMode_ShouldNotPropagate`	Do not propagate this attachment.
`kCVAttachmentMode_ShouldPropagate`	Copy this attachment when using the CVBufferPropagateAttachments function. For example, in most cases, you would want to propagate an attachment bearing a timestamp to each successive buffer.
`kCVBufferMovieTimeKey`	The movie time associated with the buffer. Generally only available for frames emitted by QuickTime (type CFDictionary containing the kCVBufferTimeValueKey and kCVBufferTimeScaleKey keys).

`kCVBufferNonPropagatedAttachmentsKey`	Attachments that should not be copied when using the CVBufferPropagateAttachments function (type CFDictionary, containing a list of attachments as key-value pairs).
`kCVBufferPropagatedAttachmentsKey`	Attachments that should be copied when using the CVBufferPropagateAttachments function (type CFDictionary, containing a list of attachments as key-value pairs).
`kCVBufferTimeScaleKey`	The time scale associated with the movie.
`kCVBufferTimeValueKey`	The actual time value associated with the movie.

CVDisplayLink.h

Functions

All of the new functions in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`CVDisplayLinkCreateWithActiveCGDisplays`	Creates a display link capable of being used with all active displays.
`CVDisplayLinkCreateWithCGDisplay`	Creates a display link for a single display.
`CVDisplayLinkCreateWithCGDisplays`	Creates a display link for an array of displays.
`CVDisplayLinkCreateWithOpenGLDisplayMask`	Creates a display link from an OpenGL display mask.
`CVDisplayLinkGetActualOutputVideoRefreshPeriod`	Retrieves the actual output refresh period of a display as measured by the host time base.
`CVDisplayLinkGetCurrentCGDisplay`	Gets the current display associated with a display link.
`CVDisplayLinkGetCurrentTime`	Retrieves the current ("now") time of a given display link.
`CVDisplayLinkGetNominalOutputVideoRefreshPeriod`	Retrieves the nominal refresh period of a display link.
`CVDisplayLinkGetOutputVideoLatency`	Retrieves the nominal latency of a display link.
`CVDisplayLinkGetTypeID`	Obtains the Core Foundation ID for the display link data type.

`CVDisplayLinkIsRunning`	Indicates whether a given display link is running.
`CVDisplayLinkRelease`	Releases a display link.
`CVDisplayLinkRetain`	Retains a display link.
`CVDisplayLinkSetCurrentCGDisplay`	Sets the current display of a display link.
`CVDisplayLinkSetCurrentCGDisplayFromOpenGLContext`	Selects the display link most optimal for the current renderer of an OpenGL context.
`CVDisplayLinkSetOutputCallback`	Set the renderer output callback function.
`CVDisplayLinkStart`	Activates a display link.
`CVDisplayLinkStop`	Stops a display link.
`CVDisplayLinkTranslateTime`	Translates the time in the display link's time base from one representation to another.

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`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.
`CVDisplayLinkRef`	Defines a display link.

CVHostTime.h

Functions

All of the new functions in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`CVGetCurrentHostTime`	Retrieves the current value of the host time base.
`CVGetHostClockFrequency`	Retrieve the frequency of the host time base.
`CVGetHostClockMinimumTimeDelta`	Retrieve the smallest possible increment in the host time base.

CVImageBuffer.h

Functions

All of the new functions in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`CVImageBufferGetCleanRect`	Returns the source rectangle of a Core Video image buffer that represents the clean aperture of the buffer in encoded pixels.
`CVImageBufferGetColorSpace`	Returns the color space of a Core Video image buffer.
`CVImageBufferGetDisplaySize`	Returns the nominal output display size, in square pixels, of a Core Video image buffer.
`CVImageBufferGetEncodedSize`	Returns the full encoded dimensions of a Core Video image buffer.

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`CVImageBufferRef`	Defines a Core Video image buffer.
`kCVImageBufferCGColorSpaceKey`	The color space for the buffer (type CGColorSpaceRef).
`kCVImageBufferCleanApertureHeightKey`	The clean aperture height (type CFNumber).
`kCVImageBufferCleanApertureHorizontalOffsetKey`	The clean aperture horizontal offset (type CFNumber).
`kCVImageBufferCleanApertureKey`	The clean aperture for the buffer (type CFDictionary , containing the clean aperture width, height, and horizontal and vertical offset key-value pairs).
`kCVImageBufferCleanApertureVerticalOffsetKey`	The clean aperture vertical offset (type CFNumber).
`kCVImageBufferCleanApertureWidthKey`	The clean aperture width (type CFNumber).
`kCVImageBufferDisplayDimensionsKey`	The buffer display dimensions (type CFDictionary containing the buffer display width and height keys).
`kCVImageBufferDisplayHeightKey`	The buffer display height (type CFNumber).

`kCVImageBufferDisplayWidthKey`	The buffer display width (type CFNumber).
`kCVImageBufferFieldCountKey`	The field count for the buffer (type CFNumber).
`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).
`kCVImageBufferFieldDetailSpatialFirstLineEarly`	(type CFString).
`kCVImageBufferFieldDetailSpatialFirstLineLate`	(type CFString).
`kCVImageBufferFieldDetailTemporalBottomFirst`	(type CFString).
`kCVImageBufferFieldDetailTemporalTopFirst`	(type CFString).
`kCVImageBufferGammaLevelKey`	The gamma level for this buffer (type CFNumber).
`kCVImageBufferPixelAspectRatioHorizontalSpacingKey`	The horizontal component of the buffer aspect ratio (type CFNumber).
`kCVImageBufferPixelAspectRatioKey`	The pixel aspect ratio of the buffer (type CFDictionary, containing the horizontal and vertical spacing keys).
`kCVImageBufferPixelAspectRatioVerticalSpacingKey`	The vertical component of the buffer aspect ratio (type CFNumber).
`kCVImageBufferPreferredCleanApertureKey`	The preferred clean aperture for the buffer (type CFDictionary , containing the clean aperture width, height, and horizontal and vertical offset key-value pairs).
`kCVImageBufferYCbCrMatrixKey`	The type of conversion matrix used for this buffer when converting from YCbCr to RGB images (type CFString).
`kCVImageBufferYCbCrMatrix_ITU_R_601_4`	Specifies the YCbCr to RGB conversion matrix for standard digital television (ITU R 601) images.
`kCVImageBufferYCbCrMatrix_ITU_R_709_2`	Specifies the YCbCr to RGB conversion matrix for HDTV digital television (ITU R 709) images.
`kCVImageBufferYCbCrMatrix_SMPTE_240M_1995`	Specifies the YCbCR to RGB conversion matrix for 1920 x 1135 HDTV (SMPTE 240M 1995).

CVOpenGLBuffer.h

Functions

All of the new functions in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`CVOpenGLBufferAttach`	Attaches an OpenGL context to a Core Video OpenGL buffer.
`CVOpenGLBufferCreate`	Create a new Core Video OpenGL buffer that can be used for OpenGL rendering purposes
`CVOpenGLBufferGetAttributes`	Obtains the attributes of a Core Video OpenGL buffer.
`CVOpenGLBufferGetTypeID`	Obtains the Core Foundation type ID for the OpenGL buffer type.
`CVOpenGLBufferRelease`	Releases a Core Video OpenGL buffer.
`CVOpenGLBufferRetain`	Retains a Core Video OpenGL buffer.

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`CVOpenGLBufferRef`	Defines a Core Video OpenGL buffer.
`kCVOpenGLBufferHeight`	The height of the buffer.
`kCVOpenGLBufferInternalFormat`	The OpenGL internal format of this buffer.
`kCVOpenGLBufferMaximumMipmapLevel`	The maximum mipmap level for this buffer.
`kCVOpenGLBufferTarget`	The OpenGL target for this buffer.
`kCVOpenGLBufferWidth`	The width of the buffer.

CVOpenGLBufferPool.h

Functions

All of the new functions in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`CVOpenGLBufferPoolCreate`	Creates a new OpenGL buffer pool.

CVOpenGLBufferPoolCreateOpenGLBuffer	Creates a new OpenGL buffer from an OpenGL buffer pool.
CVOpenGLBufferPoolGetAttributes	Returns the pool attributes dictionary for an Open GL buffer pool.
CVOpenGLBufferPoolGetOpenGLBufferAttributes	Returns the attributes of OpenGL buffers that will be created from a buffer pool.
CVOpenGLBufferPoolGetTypeID	Obtains the Core Foundation ID for the OpenGL buffer pool type.
CVOpenGLBufferPoolRelease	Releases an OpenGL buffer pool.
CVOpenGLBufferPoolRetain	Retains an OpenGL buffer pool.

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

CVOpenGLBufferPoolRef	Defines an OpenGL buffer pool.
kCVOpenGLBufferPoolMaximumBufferAgeKey	Indicates how long unused buffers should be kept before they are deallocated (type CFAbsoluteTime).
kCVOpenGLBufferPoolMinimumBufferCountKey	Indicates the minimum number of buffers to keep in the pool (type CFNumber).

CVOpenGLTexture.h

Functions

All of the new functions in this header file are listed alphabetically, with links to documentation and abstracts, if available.

CVOpenGLTextureGetCleanTexCoords	Returns the texture coordinates for the part of the image that should be displayed.
CVOpenGLTextureGetName	Returns the texture target name of a CoreVideo OpenGL texture.
CVOpenGLTextureGetTarget	Returns the texture target (for example, GL_TEXTURE_2D) of an OpenGL texture.
CVOpenGLTextureGetTypeID	Obtains the Core Foundation ID for the Core Video OpenGL texture type.
CVOpenGLTextureIsFlipped	Determines whether or not an OpenGL texture is flipped vertically.

CVOpenGLTextureRelease	Releases a Core Video OpenGL texture.
CVOpenGLTextureRetain	Retains a Core Video OpenGL texture.

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

CVOpenGLTextureRef	Defines an OpenGL texture-based image buffer.

CVOpenGLTextureCache.h

Functions

All of the new functions in this header file are listed alphabetically, with links to documentation and abstracts, if available.

CVOpenGLTextureCacheCreate	Creates an OpenGL texture cache.
CVOpenGLTextureCacheCreateTextureFromImage	Creates an OpenGL texture object from an existing image buffer.
CVOpenGLTextureCacheFlush	Flushes the OpenGL texture cache.
CVOpenGLTextureCacheGetTypeID	Returns the Core Foundation ID of the texture cache type.
CVOpenGLTextureCacheRelease	Releases an OpenGL texture cache.
CVOpenGLTextureCacheRetain	Retains an OpenGL texture cache.

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

CVOpenGLTextureCacheRef	Defines a CoreVideo OpenGL texture cache.

CVPixelBuffer.h

Functions

All of the new functions in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`CVPixelBufferCreate`	Creates a single pixel buffer for a given size and pixel format.
`CVPixelBufferCreateResolvedAttributesDictionary`	Takes an array of CFDictionary objects describing various pixel buffer attributes and tries to resolve them into a single dictionary.
`CVPixelBufferCreateWithBytes`	Creates a pixel buffer for a given size and pixel format containing data specified by a memory location.
`CVPixelBufferCreateWithPlanarBytes`	Creates a single pixel buffer in planar format for a given size and pixel format containing data specified by a memory location.
`CVPixelBufferFillExtendedPixels`	Fills the extended pixels of the pixel buffer.
`CVPixelBufferGetBaseAddress`	Returns the base address of the pixel buffer.
`CVPixelBufferGetBaseAddressOfPlane`	Returns the base address of the plane at the specified plane index.
`CVPixelBufferGetBytesPerRow`	Returns the number of bytes per row of the pixel buffer.
`CVPixelBufferGetBytesPerRowOfPlane`	Returns the number of bytes per row for a plane at the specified index in the pixel buffer.
`CVPixelBufferGetDataSize`	Returns the data size for contiguous planes of the pixel buffer.
`CVPixelBufferGetExtendedPixels`	Returns the amount of extended pixel padding in the pixel buffer.
`CVPixelBufferGetHeight`	Returns the height of the pixel buffer.
`CVPixelBufferGetHeightOfPlane`	Returns the height of the plane at planeIndex in the pixel buffer.
`CVPixelBufferGetPixelFormatType`	Returns the pixel format type of the pixel buffer.
`CVPixelBufferGetPlaneCount`	Returns number of planes of the pixel buffer.
`CVPixelBufferGetTypeID`	Returns the Core Foundation ID of the pixel buffer type.
`CVPixelBufferGetWidth`	Returns the width of the pixel buffer.

CVPixelBufferGetWidthOfPlane	Returns the width of the plane at a given index in the pixel buffer.
CVPixelBufferIsPlanar	Determine if the pixel buffer is planar.
CVPixelBufferLockBaseAddress	Locks the base address of the pixel buffer.
CVPixelBufferRelease	Releases a pixel buffer.
CVPixelBufferRetain	Retains a pixel buffer.
CVPixelBufferUnlockBaseAddress	Unlocks the base address of the pixel buffer.

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

CVPixelBufferRef	Defines a Core Video pixel buffer.
CVPixelBufferReleaseBytesCallback	Defines a pointer to a pixel buffer release callback function, which is called when a pixel buffer created by CVPixelBufferCreateWithBytes is released.
CVPixelBufferReleasePlanarBytesCallback	Defines a pointer to a pixel buffer release callback function, which is called when a pixel buffer created by CVPixelBufferCreateWithPlanarBytes is released.
kCVPixelBufferBytesPerRowAlignmentKey	Indicates the number of bytes per row in the pixel buffer (type CFNumber).
kCVPixelBufferCGBitmapContextCompatibilityKey	Indicates whether the pixel buffer is compatible with Core Graphics bitmap contexts (type CFBoolean).
kCVPixelBufferCGImageCompatibilityKey	Indicates whether the pixel buffer is compatible with CGImage types (type CFBoolean).
kCVPixelBufferExtendedPixelsBottomKey	The number of pixels padding the bottom of the image (type CFNumber).
kCVPixelBufferExtendedPixelsLeftKey	The number of pixels padding the left of the image (type CFNumber).
kCVPixelBufferExtendedPixelsRightKey	The number of pixels padding the right of the image (type CFNumber).

`kCVPixelBufferExtendedPixelsTopKey`	The number of pixels padding the top of the image (type CFNumber).
`kCVPixelBufferHeightKey`	The height of the pixel buffer (type CFNumber).
`kCVPixelBufferMemoryAllocatorKey`	The allocator used with this buffer (type CFAllocatorRef).
`kCVPixelBufferOpenGLCompatibilityKey`	Indicates whether the pixel buffer is compatible with OpenGL contexts (type CFBoolean).
`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.
`kCVPixelBufferWidthKey`	The width of the pixel buffer (type CFNumber).

CVPixelBufferPool.h

Functions

All of the new functions in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`CVPixelBufferPoolCreate`	Creates a pixel buffer pool.
`CVPixelBufferPoolCreatePixelBuffer`	Creates a pixel buffer from a pixel buffer pool.
`CVPixelBufferPoolGetAttributes`	Returns the pool attributes dictionary for a pixel buffer pool.
`CVPixelBufferPoolGetPixelBufferAttributes`	Returns the attributes of pixel buffers that will be created from this pool.
`CVPixelBufferPoolGetTypeID`	Returns the Core Foundation ID of the pixel buffer pool type.
`CVPixelBufferPoolRelease`	Releases a pixel buffer pool.
`CVPixelBufferPoolRetain`	Retains a pixel buffer pool.

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`CVPixelBufferPoolRef`	Defines a pixel buffer pool.
`kCVPixelBufferPoolMaximumBufferAgeKey`	The maximum allowable age for a buffer in the pixel buffer pool (type CFAbsoluteTime).
`kCVPixelBufferPoolMinimumBufferCountKey`	The minimum number of buffers allowed in the pixel buffer pool (type CFNumber).

CVPixelFormatDescription.h

Functions

All of the new functions in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`CVPixelFormatDescriptionArrayCreate-WithAllPixelFormatTypes`	Returns all the pixel format descriptions known to Core Video.
`CVPixelFormatDescriptionCreateWithPixelFormatType`	Creates a pixel format description from a given OSType identifier.
`CVPixelFormatDescriptionRegister-DescriptionWithPixelFormatType`	Registers a pixel format description with Core Video.

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`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.
`fillCallBack`	
`kCVPixelFormatBitsPerBlock`	The number of bits per block.
`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.

`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.
`kCVPixelFormatBlockVerticalAlignment`	The vertical alignment requirements of this format (type CFNumber). Assumed to be 1 if this key is not present.
`kCVPixelFormatBlockWidth`	The width, in pixels, of the smallest byte-addressable group of pixels (type CFNumber.
`kCVPixelFormatCGBitmapContextCompatibility`	Indicates whether this format is compatible with Core Graphics bitmap contexts(type CFBoolean).
`kCVPixelFormatCGBitmapInfo`	The Core Graphics bitmap information for this pixel format (if applicable).
`kCVPixelFormatCGImageCompatibility`	Indicates whether this format is compatible with the CGImage type (type CFBoolean).
`kCVPixelFormatCodecType`	The codec type (type CFString). For example, '2vuy' or k422YpCbCr8CodecType.
`kCVPixelFormatConstant`	The pixel format constant for QuickTime.
`kCVPixelFormatFillExtendedPixelsCallback`	Specifies a custom extended pixel fill algorithm (type CFData). See CVFillExtendedPixelsCallBack and CVFillExtendedPixelsCallbackData for more information.
`kCVPixelFormatFourCC`	The Microsoft FourCC equivalent code for this pixel format (type CFString).
`kCVPixelFormatHorizontalSubsampling`	Horizontal subsampling information for this plane (type CFNumber). Assumed to be 1 if this key is not present.
`kCVPixelFormatName`	The name of the pixel format (type CFString). This should be the same as the codec name you would use in QuickTime.
`kCVPixelFormatOpenGLCompatibility`	Indicates whether this format is compatible with OpenGL (type CFBoolean).
`kCVPixelFormatOpenGLFormat`	The OpenGL format used to describe this image plane (if applicable). See the OpenGL specification for possible values.

`kCVPixelFormatOpenGLInternalFormat`	The OpenGL internal format for this pixel format (if applicable). See the OpenGL specification for possible values.
`kCVPixelFormatOpenGLType`	The OpenGL type to describe this image plane (if applicable). See the OpenGL specification for possible values.
`kCVPixelFormatPlanes`	The number of image planes associated with this format (type CFNumber.
`kCVPixelFormatQDCompatibility`	Indicates whether this format is compatible with QuickDraw (type CFBoolean).
`kCVPixelFormatVerticalSubsampling`	Vertical subsampling information for this plane (type CFNumber). Assumed to be 1 if this key is not present.

CVReturn.h

Data Types & Constants

All of the new data types and constants in this header file are listed alphabetically, with links to documentation and abstracts, if available.

`CVReturn`	Defines the return error code for Core Video functions.
`kCVReturnAllocationFailed`	Memory allocation for a buffer or buffer pool failed.
`kCVReturnDisplayLinkAlreadyRunning`	The specified display link is already running.
`kCVReturnDisplayLinkCallbacksNotSet`	No callback registered for the specified display link. You must set either the output callback or both the render and display callbacks.
`kCVReturnDisplayLinkNotRunning`	The specified display link is not running.
`kCVReturnError`	An otherwise undefined error occurred.
`kCVReturnFirst`	Placeholder to mark the beginning of Core Video result codes (not returned by any functions).
`kCVReturnInvalidArgument`	Invalid function parameter. For example, out of range or the wrong type.
`kCVReturnInvalidDisplay`	The display specified when creating a display link is invalid.
`kCVReturnInvalidPixelBufferAttributes`	A buffer cannot be created with the specified attributes.

`kCVReturnInvalidPixelFormat`	The buffer does not support the specified pixel format.
`kCVReturnInvalidPoolAttributes`	A buffer pool cannot be created with the specified attributes.
`kCVReturnInvalidSize`	The buffer cannot support the requested buffer size (usually too big).
`kCVReturnLast`	Placeholder to mark the end of Core Video result codes (not returned by any functions).
`kCVReturnPixelBufferNotOpenGLCompatible`	The pixel buffer is not compatible with OpenGL due to an unsupported buffer size, pixel format, or attribute.
`kCVReturnPoolAllocationFailed`	Allocation for a buffer pool failed, most likely due to a lack of resources. Check to make sure your parameters are in range.
`kCVReturnSuccess`	No error

Document Revision History

This table describes the changes to *Quartz Core Reference Update*.

Date	Notes
2007-07-18	Updated with the symbols added to the Quartz Core framework in Mac OS X v10.5.
2005-07-07	Fixed links.
2005-04-29	Minor wording changes.
	New document that summarizes the symbols added to the Quartz Core framework in Mac OS X v10.4.