Quartz Core Framework Reference

2006-05-23

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

Simultaneously published in the United States and Canada.

Contents

Introduction

Framework	/System/Library/Frameworks/QuartzCore
Header file directories	/System/Library/Frameworks/QuartzCore.framework/Headers
Companion guides:	Core Image Programming Guide 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

CIColor Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding 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

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 the CIFilter, CIContext, CIVector, and CIImage classes 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, 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 16)
  Initializes a color object with a Quartz color.

Creating color objects

+ colorWithCGColor: (page 13)
  Creates a color object from a Quartz color.

+ colorWithRed:green:blue: (page 13)
  Creates a color object using the specified RGB color component values

+ colorWithRed:green:blue:alpha: (page 14)
  Creates a color object using the specified RGBA color component values.

+ colorWithString: (page 14)
  Creates a color object using the RGBA color component values specified by a string.

Getting color components

- alpha (page 15)
  Returns the alpha value of the color.

- blue (page 15)
  Returns the blue component of the color.

- colorSpace (page 15)
  Returns the Quartz 2D color space associated with the color.

- components (page 15)
  Returns the color components of the color.

- green (page 16)
  Returns the green component of the color.

- numberOfComponents (page 16)
  Returns the number of color components in the color.

- red (page 16)
  Returns the red component of the color.

- stringRepresentation (page 17)
  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`) 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` 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` 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.

colorWithRed:green:blue:

Creates a color object using the specified RGB color component values

```
+ (CIColor *)colorWithRed:(float)r green:(float)g blue:(float)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.

colorWithRed:green:blue:alpha:

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

```
+ (CIColor *)colorWithRed:(float)r green:(float)g blue:(float)b alpha:(float)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.

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.

Instance Methods

alpha

Returns the alpha value of the color.

```
- (float)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.

blue

Returns the blue component of the color.

```
- (float)blue
```

Return Value
The unpremultiplied blue component of the color.

Availability
Mac OS X v10.4 and later.

colorSpace

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

```
- (CGColorSpaceRef)colorSpace
```

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

Availability
Mac OS X v10.4 and later.

components

Returns the color components of the color.

```
- (constant float *)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.

green

Returns the green component of the color.

```
- (float)green
```

Return Value
The unpremultiplied green component of the color.

Availability
Mac OS X v10.4 and later.

initWithCGColor:

Initializes a color object with a Quartz color.

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

Parameters

c

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

Discussion
A `CGColorRef` 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` 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.

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.

red

Returns the red component of the color.

```
- (float)red
```

Return Value
The unpremultiplied red component of the color.

Availability
Mac OS X v10.4 and later.

stringRepresentation

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

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

Return Value
The formatted string.

Availability
Mac OS X v10.4 and later.

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 guide:	Core Image Programming Guide

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 CIFilter, CIImage, CIVector, and CIColor objects to take advantage of the built-in Core Image filters when processing images.

Tasks

Creating a context

`+ contextWithCGContext:options:` (page 20)
Creates a Core Image context from a Quartz context, using the specified options.

`+ contextWithCGLContext:pixelFormat:options:` (page 21)
Creates a Core Image context from a CGL context, using the specified options and pixel format object.

Rendering images

- `createCGImage:fromRect:` (page 22)
  Creates a Quartz 2D image from a region of a CIImage object.
- `createCGLayerWithSize:info:` (page 22)
  Creates a CGLayer object from the specified parameters.
- `drawImage:atPoint:fromRect:` (page 23)
  Renders a region of an image to a point in the context destination.
- `drawImage:inRect:fromRect:` (page 23)
  Renders a region of an image to a rectangle in the context destination.

Managing resources

- `clearCaches` (page 22)
  Frees any cached data, such as temporary images, associated with the context and runs the garbage collector.
- `reclaimResources` (page 24)
  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.

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

Parameters

ctx

A Quartz graphics context (`CGContextRef`) 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 24) or kCIContextWorkingColorSpace (page 25) along with a `CGColorSpaceRef` for each color space.

Discussion

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

Availability

Mac OS X v10.4 and later.

contextWithCGLContext:pixelFormat:options:

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

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

Parameters

ctx

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

pf

A CGL pixel format object (`CGLPixelFormatObj`) 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 24) or kCIContextWorkingColorSpace (page 25) along with a `CGColorSpaceRef` for each color space.

Discussion

After calling this method, Core Image draws content into the 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*.

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.

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.

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.

createCGLayerWithSize:info:

Creates a CGLayer object from the specified 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

Reserved for future use. Pass `NULL`.

Return Value
A CGLayer (`CGLayerRef`) object.

Discussion
After calling this method, Core Image draws content into the CGLayer object. Core Image creates a CGLayer 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 49)
`+ imageWithCGLayer:options:` (page 49)

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 `apply` methods.

Availability
Mac OS X v10.4 and later.

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 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 `apply` methods.

Availability

Mac OS X v10.4 and later.

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.

Constants

The following constants define keys that can be passed in an options dictionary when you create a context.

Constant	Description
`kCIContextOutput-ColorSpace`	Specifies 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.)

Constant	Description
`kCIContextWorking-ColorSpace`	Specifies a 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.)
`kCIContextUse-SoftwareRenderer`	Specifies whether or not to require the software renderer. If that associated NSNumber object is `YES`, then the software renderer is required.

CIFilter Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying NSObject (NSObject)
Framework	Library/Frameworks/QuartzCore.framework
Declared in:	QuartzCore/CIFilter.h
Availability:	Mac OS X v10.4 and later
Companion guides:	Core Image Programming Guide Core Image Filter Reference

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 the CIImage, CIContext, CIVector, CIImageAccumulator, and CIColor objects to take advantage of the built-in Core Image filters when processing images. CIFilter objects are also used along with CIKernel, CISampler, and CIFilterShape objects to create custom filters.

Tasks

Creating filters

`+ filterWithName:` (page 30)
Creates a `CIFilter` object for a specific kind of filter.

+ filterWithName:keysAndValues: (page 30)
Creates a CIFilter object for a specific kind of filter and initialized the input values.

Getting filter names

+ filterNamesInCategories: (page 29)
Gets all published filter names that match all the specified categories.

+ filterNamesInCategory: (page 29)
Gets the names of the filters in the specified category.

Registering a filter

+ registerFilterName:constructor:classAttributes: (page 31)
Publishes a custom filter that is not packaged as an image unit.

Getting filter parameters and attributes

- attributes (page 33)
Gets the attributes of a filter.

- inputKeys (page 34)
Gets the input parameters to a filter.

- outputKeys (page 34)
Gets the output parameters for a filter.

Setting default values

- setDefaults (page 34)
Sets all input values for a filter to default values.

Applying a filter

- apply:arguments:options: (page 32)
Produces a *CIImage* object by applying arguments to a kernel function and using options to control how the kernel function is evaluated.

- apply: (page 32)
Produces a CIImage object by applying a kernel function.

Getting localized names

`+ localizedNameForFilterName:` (page 31)
Gets the localized name for the specified filter name.

`+ localizedNameForCategory:` (page 30)
Gets the localized name for the specified filter category.

Class Methods

filterNamesInCategories:

Gets 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 38) and kCICategoryGenerator (page 38), 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 38), 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.

filterNamesInCategory:

Gets the names of the filters 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.

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 34) 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 30)

filterWithName:keysAndValues:

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

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

localizedNameForCategory:

Gets 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.

localizedNameForFilterName:

Gets 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.

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 35) attribute and a string that specifies the display name, and the `kCIAttributeFilterCategories` (page 35) and an array that specifies the categories to which the filter belongs (such as `kCICategoryStillImage` (page 38) and `kCICategoryDistortionEffect` (page 37)). 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.

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 32)

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 32)

attributes
Gets the attributes of a 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.

inputKeys

Gets the input parameters to a 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.

outputKeys

Gets the output parameters for a 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.

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.

Constants

The following 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.

Constant	Description
`kCIAttributeFilterName`	The filter name, specified as an NSString object. Available in Mac OS X v10.4 and later.
`kCIAttributeFilter-DisplayName`	The localized version of the filter name that is displayed in the user interface. Available in Mac OS X v10.4 and later.
`kCIAttributeFilter-Categories`	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.
`kCIAttributeClass`	The attribute class. Available in Mac OS X v10.4 and later.
`kCIAttributeType`	The attribute type. Available in Mac OS X v10.4 and later.
`kCIAttributeMin`	The minimum value for a filter parameter, specified as a floating-point value. Available in Mac OS X v10.4 and later.
`kCIAttributeMax`	The maximum value for a filter parameter, specified as a floating-point value. Available in Mac OS X v10.4 and later.
`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.
`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.

Constant	Description
kCIAttributeDefault	The default value, specified as a floating-point value, for a filter parameter. Available in Mac OS X v10.4 and later.
kCIAttributeIdentity	A value that results in no effect on the input image. Available in Mac OS X v10.4 and later.
kCIAttributeName	The name of the attribute. Available in Mac OS X v10.4 and later.
kCIAttributeDisplayName	The localized display name of the attribute. Available in Mac OS X v10.4 and later.

The following are attribute keys used for numeric data types.

Constant	Description
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.
kCIAttributeTypeScalar	A scalar value. Available in Mac OS X v10.4 and later.
kCIAttributeTypeDistance	A distance. Available in Mac OS X v10.4 and later.
kCIAttributeTypeAngle	An angle. Available in Mac OS X v10.4 and later.
kCIAttributeTypeBoolean	A Boolean value. Available in Mac OS X v10.4 and later.

The following are attribute keys used for vector quantities.

Constant	Description
kCIAttributeTypePosition	A two-dimensional location in the working coordinate space. (A 2-element vector type.) Available in Mac OS X v10.4 and later.
kCIAttributeTypeOffset	An offset. (A 2-element vector type.) Available in Mac OS X v10.4 and later.
kCIAttributeType-Position3	A three-dimensional location in the working coordinate space. (A 3-element vector type.) Available in Mac OS X v10.4 and later.

Constant	Description
`kCIAttributeType-Rectangle`	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.

The following is an attribute key used for color.

Constant	Description
`kCIAttributeType-OpaqueColor`	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.

The following is an attribute key used for gradient images.

Constant	Description
`kCIAttributeTypeGradient`	An n-by-1 gradient image used to describe a color ramp. Available in Mac OS X v10.4 and later.

The following are keys used for filter categories.

Constant	Description
`kCICategoryDistortionEffect`	A distortion effect. Available in Mac OS X v10.4 and later.
`kCICategoryGeometryAdjustment`	Geometry adjustment. Available in Mac OS X v10.4 and later.
`kCICategoryCompositeOperation`	A compositing effect. Available in Mac OS X v10.4 and later.
`kCICategoryHalftoneEffect`	A halftone effect. Available in Mac OS X v10.4 and later.
`kCICategoryColorAdjustment`	Color adjustment. Available in Mac OS X v10.4 and later.
`kCICategoryColorEffect`	A color effect. Available in Mac OS X v10.4 and later.
`kCICategoryTransition`	A transition from one image to another. Available in Mac OS X v10.4 and later.
`kCICategoryTileEffect`	A tile effect. Available in Mac OS X v10.4 and later.

Constant	Description
kCICategoryGenerator	A generated image. Available in Mac OS X v10.4 and later.
kCICategoryGradient	A generated gradient image. Available in Mac OS X v10.4 and later.
kCICategoryStylize	A stylize effect. Available in Mac OS X v10.4 and later.
kCICategorySharpen	Sharpening filter. Available in Mac OS X v10.4 and later.
kCICategoryBlur	A blurring effect. Available in Mac OS X v10.4 and later.
kCICategoryVideo	An effect that can work on video images. Available in Mac OS X v10.4 and later.
kCICategoryStillImage	An effect that can work on still images. Available in Mac OS X v10.4 and later.
kCICategoryInterlaced	An effect that can work on interlaced images. Available in Mac OS X v10.4 and later.
kCICategoryNonSquarePixels	An effect that can work on non-square pixels. Available in Mac OS X v10.4 and later.
kCICategoryHighDynamicRange	An effect that can work on high dynamic range pixels. Available in Mac OS X v10.4 and later.
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.

The following are keys used as options to the `apply` method.

Constant	Description
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. Available in Mac OS X v10.4 and later.
kCIApplyOption-Definition	Specifies 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.
kCIApplyOption-UserInfo	Specifies to retain the associated object and pass it to any callbacks invoked for that filter. Available in Mac OS X v10.4 and later.

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

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 the CIFilter, CIKernel, and CISampler objects to create custom filters.

Tasks

Creating a filter shape

`+ shapeWithRect:` (page 40)
Creates a filter shape object and initializes it with a rectangle.

Initializing a filter shape

`- initWithRect:` (page 40)
Initializes a filter shape object with a rectangle.

Modifying a filter shape

- `insetByX:Y:` (page 41)
  Modifies a filter shape object so that it is inset by the specified x and y values.
- `intersectWith:` (page 41)
  Creates a filter shape object that represents the intersection of the current filter shape and the specified filter shape object.
- `intersectWithRect:` (page 41)
  Creates a filter shape that represents the intersection of the current filter shape and a rectangle.
- `transformBy:interior:` (page 42)
  Creates a filter shape that results from applying a transform to the current filter shape.
- `unionWith:` (page 42)
  Creates a filter shape that results from the union of the current filter shape and another filter shape object.
- `unionWithRect:` (page 42)
  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.

```
+ 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.

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.

insetByX:Y:

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

```
- 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.

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.

intersectWithRect:

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

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

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.

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.

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.

unionWithRect:

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

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

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.

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

Overview

The `CIImage` class represents an image. Core Image images are immutable. You use CIImage objects in conjunction with CIFilter, CIContext, CIVector, and CIColor objects 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, URLs, and NSData.

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.

Tasks

Creating an image

- `imageByApplyingTransform:` (page 54)
  Transforms an image object by applying an affine transform.

+ `imageWithBitmapData:bytesPerRow:size:format:colorSpace:` (page 47)
  Creates an image object from bitmap data.

+ `imageWithCGImage:` (page 48)
  Creates an image object from a Quartz 2D image.

+ `imageWithCGImage:options:` (page 48)
  Creates an image object from a Quartz 2D image using the specified options.

+ `imageWithCGLayer:` (page 49)
  Creates an image object from the contents supplied by a CGLayer object.

+ `imageWithCGLayer:options:` (page 49)
  Creates an image object from the contents supplied by a CGLayer object, using the specified options.

+ `imageWithContentsOfURL:` (page 50)
  Creates an image object from the contents of a file.

+ `imageWithContentsOfURL:options:` (page 50)
  Creates an image object from the contents of a file, using the specified options.

+ `imageWithCVImageBuffer:` (page 51)
  Creates an image object from the contents of CVImageBuffer object.

+ `imageWithCVImageBuffer:options:` (page 51)
  Creates an image object from the contents of CVImageBuffer object, using the specified options.

+ `imageWithData:` (page 51)
  Creates an image object initialized with the supplied image data.

+ `imageWithData:options:` (page 52)
  Creates an image object initialized with the supplied image data, using the specified options.

+ `imageWithImageProvider:size::format:colorSpace:options:` (page 52)
  Creates an image object initialized with data provided by an image provider.

+ `imageWithTexture:size:flipped:colorSpace:` (page 53)
  Creates an image object initialized with data supplied by an OpenGL texture.

Initializing an image

- `initWithBitmapData:bytesPerRow:size:format:colorSpace:` (page 54)
  Initializes an image object with bitmap data.

- `initWithCGImage:` (page 55)
  Initializes an image object with a Quartz 2D image.
- `initWithCGImage:options:` (page 55)
  Initializes an image object with a Quartz 2D image, using the specified options.
- `initWithCGLayer:` (page 56)
  Initializes an image object from the contents supplied by a CGLayer object.
- `initWithCGLayer:options:` (page 56)
  Initializes an image object from the contents supplied by a CGLayer object, using the specified options.
- `initWithContentsOfURL:` (page 57)
  Initializes an image object from the contents of a file.
- `initWithContentsOfURL:options:` (page 57)
  Initializes an image object from the contents of a file, using the specified options.
- `initWithCVImageBuffer:` (page 57)
  Initializes an image object from the contents of CVImageBuffer object.
- `initWithCVImageBuffer:options:` (page 58)
  Initializes an image object from the contents of CVImageBuffer object, using the specified options.
- `initWithData:` (page 58)
  Initializes an image object with the supplied image data.
- `initWithData:options:` (page 59)
  Initializes an image object with the supplied image data, using the specified options.
- `initWithImageProvider:size::format:colorSpace:options:` (page 59)
  Initializes an image object with data provided by an image provider, using the specified options.
- `initWithTexture:size:flipped:colorSpace:` (page 60)
  Initializes an image object with data supplied by an OpenGL texture.

Getting image information

- `definition` (page 54)
  Gets the domain of definition of an image.
- `extent` (page 54)
  Gets the extent of an image.

Class Methods

imageWithBitmapData:bytesPerRow:size:format:colorSpace:

Creates 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 constants, either kCIFormatARGB8 (page 61) (32 bit-per-pixel, fixed-point pixel format) or kCIFormatRGBAf (page 61) (128 bit-per-pixel, floating-point pixel format).

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.

imageWithCGImage:

Creates 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.

imageWithCGImage:options:

Creates an image object from a Quartz 2D image using the specified options.

```
+ (CIImage *)imageWithCGImage:(CGImageRef)image options:(NSDictionary *)d
```

Parameters

image

A Quartz 2D image (`CGImageRef`) object. For more information, see *Quartz 2D Programming Guide* and *CGImage Reference*.

d

A dictionary that contains options for creating an image object. You can supply such options as a pixel format and a color space (kCIImageColorSpace (page 61)). The pixel format must be a CIFormat (page 61) data type, either kCIFormatARGB8 (page 61) or kCIFormatRGBAf (page 61).

Return Value

An image object initialized with the contents of the Quartz 2D image and set up with the specified options.

Availability

Mac OS X v10.4 and later.

imageWithCGLayer:

Creates 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 CGLayer object.

Availability

Mac OS X v10.4 and later.

See Also

`- createCGLayerWithSize:info:` (page 22)

imageWithCGLayer:options:

Creates 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 (kCIImageColorSpace (page 61)). The pixel format must be a CIFormat (page 61) data type, either kCIFormatARGB8 (page 61) or kCIFormatRGBAf (page 61).

Return Value
An image object initialized with the contents of the CGLayer object and set up with the specified options.

Availability
Mac OS X v10.4 and later.

See Also
`- createCGLayerWithSize:info:` (page 22)

imageWithContentsOfURL:

Creates 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.

imageWithContentsOfURL:options:

Creates 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 (kCIImageColorSpace (page 61)). The pixel format must be a CIFormat (page 61) data type, either kCIFormatARGB8 (page 61) or kCIFormatRGBAf (page 61).

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.

imageWithCVImageBuffer:

Creates 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 CVImageBuffer object.

Availability

Mac OS X v10.4 and later.

imageWithCVImageBuffer:options:

Creates 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 pixel format and a color space (kCIImageColorSpace (page 61)). The pixel format must be a CIFormat (page 61) data type, either kCIFormatARGB8 (page 61) or kCIFormatRGBAf (page 61).

Return Value

An image object initialized with the contents of the CVImageBuffer object and set up with the specified options.

Availability

Mac OS X v10.4 and later.

imageWithData:

Creates an image object initialized with the supplied image data.

```
+ (CIImage *)imageWithData: (NSData *)data
```

Parameters

data

A pointer to the image data. The data must be premultiplied.

Return Value

An image object initialized with the supplied data.

Availability
Mac OS X v10.4 and later.

imageWithData:options:

Creates 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 (kCIImageColorSpace (page 61)). The pixel format must be a CIFormat (page 61) data type, either kCIFormatARGB8 (page 61) or kCIFormatRGBAf (page 61).

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.

imageWithImageProvider:size::format:colorSpace:options:

Creates 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` protocol. Core Image retains this data until the image is deallocated.

The size of the image.

height

The height of the image.

f

A pixel format constant , either kCIFormatARGB8 (page 61) (32 bit-per-pixel, fixed-point pixel format) or kCIFormatRGBAf (page 61) (128 bit-per-pixel, floating-point pixel format).

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.

See Also

`- provideImageData:bytesPerRow:origin::size::userInfo:` (page 91)

imageWithTexture:size:flipped:colorSpace:

Creates an image object initialized with data supplied by an OpenGL texture.

```
+ (CIImage *)imageWithTexture:(unsigned long)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 in the CGLContext 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.

Availability
Mac OS X v10.4 and later.

Instance Methods

definition

Gets the domain of definition of an image.

```
- (CIFilterShape *)definition
```

Return Value
A filter shape object.

Availability
Mac OS X v10.4 and later.

extent

Gets the extent of an 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.

imageByApplyingTransform:

Transforms an image object by 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.

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 , either kCIFormatARGB8 (page 61) (32 bit-per-pixel, fixed-point pixel format) or kCIFormatRGBAf (page 61) (128 bit-per-pixel, floating-point pixel format).

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.

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.

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 (kCIImageColorSpace (page 61)). The pixel format must be a CIFormat (page 61) data type, either kCIFormatARGB8 (page 61) or kCIFormatRGBAf (page 61).

Return Value

The initialized image object or `nil` if the object could not be initialized.

Availability

Mac OS X v10.4 and later.

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

`- createCGLayerWithSize:info:` (page 22)

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 (kCIImageColorSpace (page 61)). The pixel format must be a CIFormat (page 61) data type, either kCIFormatARGB8 (page 61) or kCIFormatRGBAf (page 61).

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
`- createCGLayerWithSize:info:` (page 22)

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.

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 (kCIImageColorSpace (page 61)). The pixel format must be a CIFormat (page 61) data type, either kCIFormatARGB8 (page 61) or kCIFormatRGBAf (page 61).

Return Value
The initialized image object or `nil` if the object could not be initialized.

Availability
Mac OS X v10.4 and later.

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.

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 pixel format and a color space (kCIImageColorSpace (page 61)). The pixel format must be a CIFormat (page 61) data type, either kCIFormatARGB8 (page 61) or kCIFormatRGBAf (page 61).

Return Value

The initialized image object or `nil` if the object could not be initialized.

Availability

Mac OS X v10.4 and later.

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.

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 (kCIImageColorSpace (page 61)). The pixel format must be a CIFormat (page 61) data type, either kCIFormatARGB8 (page 61) or kCIFormatRGBAf (page 61).

Return Value

The initialized image object or `nil` if the object could not be initialized.

Availability

Mac OS X v10.4 and later.

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` protocol. Core Image retains this data until the image is deallocated.

The size of the image data.

height

The height of the image data.

f

A pixel format constant , either kCIFormatARGB8 (page 61) (32 bit-per-pixel, fixed-point pixel format) or kCIFormatRGBAf (page 61) (128 bit-per-pixel, floating-point pixel format).

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.

initWithTexture:size:flipped:colorSpace:

Initializes an image object with data supplied by an OpenGL texture.

```
- initWithTexture:(unsigned long)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 in the CGLContext 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.

Availability
Mac OS X v10.4 and later.

Constants

The following constants define pixel formats.

Constant	Description
`CIFormat`	Specifies the data type for a pixel format.
`kCIFormatARGB8`	Specifies a 32 bit-per-pixel, fixed-point pixel format.
`kCIFormatRGBA16`	Specifies a 64 bit-per-pixel, fixed-point pixel format.
`kCIFormatRGBAf`	Specifies a 128 bit-per-pixel, floating-point pixel format.

The following constant specifies a key that you use to override the implicit color space of an image.

Constant	Description
`kCIImageColorSpace`	Specifies a color space for an image. 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.

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

Overview

The `CIImageAccumulator` class enables feedback-based image processing for such things as the iterative painting operations or fluid dynamics simulations. You use CIImageAccumulator objects in conjunction with CIFilter, CIContext, CIVector, and CIImage objects to take advantage of the built-in Core Image filters when processing images.

Tasks

Creating and initializing an image accumulator

`+ imageAccumulatorWithExtent:format:` (page 64)
Creates an image accumulator with the specified extent and pixel format.

`- initWithExtent:format:` (page 65)
Initializes an image accumulator with the specified extent and pixel format.

Setting an image

- `setImage:` (page 66)
  Sets the contents of the image accumulator to the contents of the specified image object.
- `setImage:dirtyRect:` (page 66)
  Set the the contents of the image accumulator to the contents of the specified image object and specifies a changed region.

Obtaining data from an image accumulator

- `extent` (page 65)
  Returns the extent of the image associated with the image accumulator.
- `format` (page 65)
  Returns the pixel format of the image accumulator.
- `image` (page 65)
  Returns the current contents of the image accumulator.

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, either `kCIFormatARGB8` (32 bit-per-pixel, fixed-point pixel format) or `kCIFormatRGBAf` (128 bit-per-pixel, floating-point pixel format).

Return Value

The image accumulator object.

Availability

Mac OS X v10.4 and later.

Instance Methods

extent

Returns the extent of the image associated with the image accumulator.

```
- (CGRect)extent
```

Return Value
The rectangle hat 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.

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.

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.

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, either `kCIFormatARGB8` (32 bit-per-pixel, fixed-point pixel format) or `kCIFormatRGBAf` (128 bit-per-pixel, floating-point pixel format).

Return Value
The initialized image accumulator object.

Availability
Mac OS X v10.4 and later.

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.

setImage:dirtyRect:

Set the the contents of the image accumulator to the contents of the specified image object and specifies a changed region.

```
- (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 changed region. You must guarantee that the new contents differ from the old only within the region specified by the this argument.

Availability
Mac OS X v10.4 and later.

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

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 the CIFIlter, CIFIlterShape, and CISampler objects to create custom filters.

Tasks

Creating a kernel

`+ kernelsWithString:` (page 68)
Creates a CIKernel object that contains one or more kernel routines.

Getting a kernel name

`- name` (page 68)
Gets the name of a kernel routine.

Setting a selector

`- setROISelector:` (page 68)
Sets the selector used by the kernel to query the region of interest of the kernel.

Class Methods

kernelsWithString:

Creates a CIKernel object that contains one or more kernel routines.

```
+ (CIKernel *)kernelsWithString:(NSString *)s
```

Parameters

s
A string object that specifies one or more functions in the Core Image dialect of the OpenGL Shading Language. Each function must be marked as a kernel.

Discussion

See *Core Image Kernel Language Reference*.

Return Value

The CIKernel object that contains the kernel functions specified by a string object..

Availability

Mac OS X v10.4 and later.

Instance Methods

name

Gets the name of a kernel routine.

```
- (NSString *)name
```

Return Value

The name of the kernel routine.

Availability

Mac OS X v10.4 and later.

setROISelector:

Sets the selector used by the kernel 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.

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 guide:	Core Image Programming Guide

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 72)
Scans directories for files that have the `.plugin` extension and then loads the image units.

`+ loadNonExecutablePlugIns` (page 72)
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 72)
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.

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.

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.

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

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 the CIFilter, CIKernel, and CIFilterShape objects to create custom filters.

Tasks

Creating a sampler

`+ samplerWithImage:` (page 76)
Creates a sampler that references an image.

`+ samplerWithImage:keysAndValues:` (page 76)
Creates a sampler that references an image using options specified as key-value pairs.

`+ samplerWithImage:options:` (page 77)
Creates a sampler that references an image using options specified in a dictionary.

Initializing a sampler

- `initWithImage:` (page 78)
  Initializes a sampler with an image object.
- `initWithImage:keysAndValues:` (page 78)
  Initializes the sampler with an image object using options specified as key-value pairs.
- `initWithImage:options:` (page 78)
  Initializes the sampler with an image object using options specified in a dictionary.

Getting Information About the Sampler Object

- `definition` (page 77)
  Gets the domain of definition (DOD) of the sampler.
- `extent` (page 78)
  Gets the rectangle that specifies the extent of the sampler.

Class Methods

samplerWithImage:

Creates 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.

samplerWithImage:keysAndValues:

Creates 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. The supported keys are kCISamplerAffineMatrix (page 79), kCISamplerWrapMode (page 79), and kCISamplerFilterMode (page 79).

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.

samplerWithImage:options:

Creates 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. The supported keys are kCISamplerAffineMatrix (page 79), kCISamplerWrapMode (page 79), and kCISamplerFilterMode (page 79).

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.

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.

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.

initWithImage:

Initializes a sampler with an image object.

```
- initWithImage:(CIImage *)im
```

Parameters

im

The image object to initialize the sampler with.

Availability

Mac OS X v10.4 and later.

initWithImage:keysAndValues:

Initializes the sampler with an image object using options specified as key-value pairs.

```
- 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. The supported keys are kCISamplerAffineMatrix (page 79), kCISamplerWrapMode (page 79), and kCISamplerFilterMode (page 79).

Availability

Mac OS X v10.4 and later.

initWithImage:options:

Initializes the sampler with an image object using options specified in a dictionary.

```
- 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. The supported keys are kCISamplerAffineMatrix (page 79), kCISamplerWrapMode (page 79), and kCISamplerFilterMode (page 79).

Availability
Mac OS X v10.4 and later.

Constants

These constants define keys that you can pass when you create a sampler.

Constant	Description
`kCISamplerAffine-Matrix`	Specifies 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.
`kCISamplerWrapMode`	Specifies how Core Image produces pixels that are outside the extent of the sample. Possible values are `kCISamplerWrapBlack` (pixels are transparent black) and `kCISamplerWrapClamp` (coordinates are clamped to the extent). Available in Mac OS X v10.4 and later.
`kCISamplerFilterMode`	Specifies the filter to use when sampling the image. Possible values are `kCISamplerFilterNearest`, for nearest neighbor sampling, and `kCISamplerFilterLinear`, for bilinear interpolation. Available in Mac OS X v10.4 and later.

These constants are values for sampler keys:

Constant	Description
`kCISamplerWrapBlack`	Specifies that pixels are transparent black. Available in Mac OS X v10.4 and later.
`kCISamplerWrapClamp`	Specifies that coordinates are clamped to the extent. Available in Mac OS X v10.4 and later.
`kCISamplerFilterNearest`	Specifies to use nearest neighbor sampling. Available in Mac OS X v10.4 and later.
`kCISamplerFilterLinear`	Specifies to use bilinear interpolation. Available in Mac OS X v10.4 and later.

CIVector Class Reference

Inherits from:	NSObject
Conforms to:	NSCopying NSCoding 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

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 the CIFilter, CIContext, CIImage, and CIColor objects to process images using the Core Image framework.

Tasks

Creating a vector

`+ vectorWithValues:count:` (page 83)
Creates a vector with a specified number of values.

`+ vectorWithX:` (page 83)
Creates a vector that contains one value.

`+ vectorWithX:Y:` (page 84)
Creates a vector that contains two values.

+ vectorWithX:Y:Z: (page 84)
Creates a vector that contains three values.

+ vectorWithX:Y:Z:W: (page 85)
Creates and initializes vector that contains four values.

+ vectorWithString: (page 83)
Creates a vector from a string.

Initializing a vector

- initWithValues:count: (page 85)
Initializes a vector with specific values.

- initWithX: (page 86)
Initializes the first position of a vector.

- initWithX:Y: (page 86)
Initializes the first two positions of a vector.

- initWithX:Y:Z: (page 86)
Initializes the first three positions of a vector.

- initWithX:Y:Z:W: (page 87)
Initializes four positions of a vector.

Getting values from a vector

- valueAtIndex: (page 87)
Gets a value from a specific position in a vector.

- count (page 85)
Gets the number of items in a vector.

- X (page 88)
Gets the value located in the first position in a vector.

- Y (page 88)
Gets the value located in the second position in a vector.

- Z (page 88)
Gets the value located in the third position in a vector.

- W (page 88)
Gets the value located in the fourth position in a vector.

- stringRepresentation (page 87)
Obtains a string representation for a vector.

Class Methods

vectorWithString:

Creates a vector from a string.

```
+ (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 87)

vectorWithValues:count:

Creates a vector with a specified number of values.

```
+ (CIVector *)vectorWithValues:(const float *)values count:(size_t)count
```

Parameters

values
The values to initialize the vector with.

count
The number of values in the vector.

Availability

Mac OS X v10.4 and later.

vectorWithX:

Creates a vector that contains one value.

```
+ (CIVector *)vectorWithX:(float)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.

vectorWithX:Y:

Creates a vector that contains two values.

```
+ (CIVector *)vectorWithX:(float)x Y:(float)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.

vectorWithX:Y:Z:

Creates a vector that contains three values.

```
+ (CIVector *)vectorWithX:(float)x Y:(float)y Z:(float)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.

vectorWithX:Y:Z:W:

Creates and initializes vector that contains four values.

```
+ (CIVector *)vectorWithX:(float)x Y:(float)y Z:(float)z W:(float)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.

Instance Methods

count

Gets 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.

initWithValues:count:

Initializes a vector with specific values.

```
- initWithValues:(const float *)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.

initWithX:

Initializes the first position of a vector.

```
- initWithX:(float)x
```

Parameters

x
The initialization value.

Availability
Mac OS X v10.4 and later.

initWithX:Y:

Initializes the first two positions of a vector.

```
- initWithX:(float)x Y:(float)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.

initWithX:Y:Z:

Initializes the first three positions of a vector.

```
- initWithX:(float)x Y:(float)y Z:(float)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.

initWithX:Y:Z:W:

Initializes four positions of a vector.

```
- initWithX:(float)x Y:(float)y Z:(float)z W:(float)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.

stringRepresentation

Obtains 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.

Availability
Mac OS X v10.4 and later.

valueAtIndex:

Gets a value from a specific position in a vector.

```
- (float)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.

W

Gets the value located in the fourth position in a vector.

```
- (float)W
```

Return Value
The value retrieved from the vector.

Availability
Mac OS X v10.4 and later.

X

Gets the value located in the first position in a vector.

```
- (float)X
```

Return Value
The value retrieved from the vector.

Availability
Mac OS X v10.4 and later.

Y

Gets the value located in the second position in a vector.

```
- (float)Y
```

Return Value
The value retrieved from the vector.

Availability
Mac OS X v10.4 and later.

Z

Gets the value located in the third position in a vector.

```
- (float)Z
```

Return Value
The value retrieved from the vector.

Availability
Mac OS X v10.4 and later.

Protocols

CIImageProvider Protocol Reference

(informal protocol)

Adopted by:	CIImage (informal protocol)
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` protocol is an informal protocol that supplies bitmap data that can be used to create or initialize a CIImage object.

Tasks

Providing image data

`- provideImageData:bytesPerRow:origin::size::userInfo:` (page 91)
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:` (page 52) to create a CIImage object from image data
- `initWithImageProvider:size::format:colorSpace:options:` (page 59) 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

The following constants specify options you can supply to an image provider through the `options` argument of `imageWithImageProvider:size::format:colorSpace:options:` (page 52) or `initWithImageProvider:size::format:colorSpace:options:` (page 59):

Constant	Description
`kCIImageProviderTileSize`	The dimensions of the image tiles requested from the image provider, specified as `NSNumber` values in an `NSArray`. Available in Mac OS X v10.4 and later.
`kCIImageProviderUserInfo`	The object passed to the image provider in the `info` argument. Available in Mac OS X v10.4 and later.

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 guide:	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 95)
Loads and initializes an image unit.

Instance Methods

load:

Loads and initializes an image unit.

```
- (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.

Other References

PART III

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

Overview

What is Core Video?

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

Who Should Read This Document

The audience for this document is any Carbon or Cocoa developer who wants a greater degree of control in manipulating video images. Developers should be familiar with digital video and OpenGL as well as multithreaded programming.

Note that most developers who merely want to display video in their applications do not need the power of Core Video; in many cases, the features in the QTKit framework or HIMovieView are sufficient.

See Also

Apple offers the following additional resources that complement the *Core Video Reference*:

- Core Video Programming Guide describes the concepts behind Core Video and contains examples of how to obtain and manipulate video frames.
- Macintosh OpenGL Programming Guide provides information on GL textures and CGL graphics contexts.
- Core Image Programming Guide contains information about how to create Core Image filters, which you can apply to video frames.

In addition, the OpenGL website (http://www.opengl.org) is the primary source for information about the OpenGL API.

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 106)
Returns a specific attachment of a Core Video buffer.

`CVBufferGetAttachments` (page 107)
Returns all attachments of a Core Video buffer.

`CVBufferPropagateAttachments` (page 108)
Copies all propagatable attachments from one Core Video buffer to another.

`CVBufferRelease` (page 108)
Releases a Core Video buffer.

`CVBufferRemoveAllAttachments` (page 108)
Removes all attachments of a Core Video buffer.

`CVBufferRemoveAttachment` (page 109)
Removes a specific attachment of a Core Video buffer.

`CVBufferRetain` (page 109)
Retains a Core Video buffer.

`CVBufferSetAttachment` (page 110)
Sets or adds an attachment of a Core Video buffer.

`CVBufferSetAttachments` (page 111)
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 112)
Creates a display link for a single display.

`CVDisplayLinkCreateWithActiveCGDisplays` (page 111)
Creates a display link capable of being used with all active displays.

`CVDisplayLinkCreateWithCGDisplays` (page 112)
Creates a display link for an array of displays.

`CVDisplayLinkCreateWithOpenGLDisplayMask` (page 113)
Creates a display link from an OpenGL display mask.

`CVDisplayLinkGetActualOutputVideoRefreshPeriod` (page 113)
Retrieves the actual output refresh period of a display as measured by the host time base.

`CVDisplayLinkGetCurrentCGDisplay` (page 114)
Gets the current display associated with a display link.

`CVDisplayLinkGetCurrentTime` (page 114)
Retrieves the current (“now”) time of a given display link.

`CVDisplayLinkGetNominalOutputVideoRefreshPeriod` (page 115)
Retrieves the nominal refresh period of a display link.

`CVDisplayLinkGetOutputVideoLatency` (page 115)
Retrieves the nominal latency of a display link.

`CVDisplayLinkGetTypeID` (page 116)
Obtains the Core Foundation ID for the display link data type.

`CVDisplayLinkIsRunning` (page 116)
Indicates whether a given display link is running.

`CVDisplayLinkRelease` (page 117)
Releases a display link.

`CVDisplayLinkRetain` (page 117)
Retains a display link.

`CVDisplayLinkSetCurrentCGDisplay` (page 117)
Sets the current display of a display link.

CVDisplayLinkSetCurrentCGDisplayFromOpenGLContext (page 118)
Selects the display link most optimal for the current renderer of an OpenGL context.

CVDisplayLinkSetOutputCallback (page 119)
Set the renderer output callback function.

CVDisplayLinkStart (page 119)
Activates a display link.

CVDisplayLinkStop (page 120)
Stops a display link.

CVDisplayLinkTranslateTime (page 120)
Translates the time in the display link's time base from one representation to another.

CVHostTime Functions

CVGetCurrentHostTime (page 121)
Retrieves the current value of the host time base.

CVGetHostClockFrequency (page 121)
Retrieve the frequency of the host time base.

CVGetHostClockMinimumTimeDelta (page 121)
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 122)
Returns the source rectangle of a Core Video image buffer that represents the clean aperture of the buffer in encoded pixels.

CVImageBufferGetColorSpace (page 122)
Returns the color space of a Core Video image buffer.

CVImageBufferGetDisplaySize (page 123)
Returns the nominal output display size, in square pixels, of a Core Video image buffer.

CVImageBufferGetEncodedSize (page 123)
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 124)
Attaches an OpenGL context to a Core Video OpenGL buffer.

CVOpenGLBufferCreate (page 124)
Create a new Core Video OpenGL buffer that can be used for OpenGL rendering purposes

CVOpenGLBufferGetAttributes (page 125)
Obtains the attributes of a Core Video OpenGL buffer.

CVOpenGLBufferGetTypeID (page 126)
Obtains the Core Foundation type ID for the OpenGL buffer type.

CVOpenGLBufferRelease (page 129)
Releases a Core Video OpenGL buffer.

CVOpenGLBufferRetain (page 130)
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 126)
Creates a new OpenGL buffer pool.

CVOpenGLBufferPoolCreateOpenGLBuffer (page 127)
Creates a new OpenGL buffer from an OpenGL buffer pool.

CVOpenGLBufferPoolGetAttributes (page 127)
Returns the pool attributes dictionary for an Open GL buffer pool.

CVOpenGLBufferPoolGetOpenGLBufferAttributes (page 128)
Returns the attributes of OpenGL buffers that will be created from a buffer pool.

CVOpenGLBufferPoolGetTypeID (page 128)
Obtains the Core Foundation ID for the OpenGL buffer pool type.

CVOpenGLBufferPoolRelease (page 128)
Releases an OpenGL buffer pool.

CVOpenGLBufferPoolRetain (page 129)
Retains an OpenGL buffer pool.

CVOpenGLTexture Functions

The Core Video OpenGL texture is a wrapper around the standard OpenGL texture type.

CVOpenGLTextureGetCleanTexCoords (page 133)
Returns the texture coordinates for the part of the image that should be displayed.

CVOpenGLTextureGetName (page 134)
Returns the texture target name of a CoreVideo OpenGL texture.

CVOpenGLTextureGetTarget (page 135)
Returns the texture target (for example, GL_TEXTURE_2D) of an OpenGL texture.

CVOpenGLTextureGetTypeID (page 135)
Obtains the Core Foundation ID for the Core Video OpenGL texture type.

CVOpenGLTextureIsFlipped (page 135)
Determines whether or not an OpenGL texture is flipped vertically.

CVOpenGLTextureRelease (page 136)
Releases a Core Video OpenGL texture.

CVOpenGLTextureRetain (page 136)
Retains a Core Video OpenGL texture.

CVOpenGLTextureCache Functions

CVOpenGLTextureCacheCreate (page 130)
Creates an OpenGL texture cache.

CVOpenGLTextureCacheCreateTextureFromImage (page 131)
Creates an OpenGL texture object from an existing image buffer.

CVOpenGLTextureCacheFlush (page 132)
Flushes the OpenGL texture cache.

CVOpenGLTextureCacheGetTypeID (page 132)
Returns the Core Foundation ID of the texture cache type.

CVOpenGLTextureCacheRelease (page 133)
Releases an OpenGL texture cache.

CVOpenGLTextureCacheRetain (page 133)
Retains an OpenGL texture cache.

CVPixelBuffer Functions

A pixel buffer stores an image in main memory

CVPixelBufferCreate (page 137)
Creates a single pixel buffer for a given size and pixel format.

CVPixelBufferCreateResolvedAttributesDictionary (page 138)
Takes an array of CFDictionary objects describing various pixel buffer attributes and tries to resolve them into a single dictionary.

CVPixelBufferCreateWithBytes (page 138)
Creates a pixel buffer for a given size and pixel format containing data specified by a memory location.

CVPixelBufferCreateWithPlanarBytes (page 140)
Creates a single pixel buffer in planar format for a given size and pixel format containing data specified by a memory location.

CVPixelBufferFillExtendedPixels (page 141)
Fills the extended pixels of the pixel buffer.

CVPixelBufferGetBaseAddress (page 142)
Returns the base address of the pixel buffer.

CVPixelBufferGetBaseAddressOfPlane (page 142)
Returns the base address of the plane at the specified plane index.

CVPixelBufferGetBytesPerRow (page 143)
Returns the number of bytes per row of the pixel buffer.

CVPixelBufferGetBytesPerRowOfPlane (page 143)
Returns the number of bytes per row for a plane at the specified index in the pixel buffer.

CVPixelBufferGetDataSize (page 143)
Returns the data size for contiguous planes of the pixel buffer.

CVPixelBufferGetExtendedPixels (page 144)
Returns the amount of extended pixel padding in the pixel buffer.

CVPixelBufferGetHeight (page 145)
Returns the height of the pixel buffer.

CVPixelBufferGetHeightOfPlane (page 145)
Returns the height of the plane at planeIndex in the pixel buffer.

CVPixelBufferGetPixelFormatType (page 145)
Returns the pixel format type of the pixel buffer.

CVPixelBufferGetPlaneCount (page 146)
Returns number of planes of the pixel buffer.

CVPixelBufferGetTypeID (page 146)
Returns the Core Foundation ID of the pixel buffer type.

CVPixelBufferGetWidth (page 147)
Returns the width of the pixel buffer.

CVPixelBufferGetWidthOfPlane (page 147)
Returns the width of the plane at a given index in the pixel buffer.

CVPixelBufferIsPlanar (page 147)
Determine if the pixel buffer is planar.

CVPixelBufferLockBaseAddress (page 148)
Locks the base address of the pixel buffer.

CVPixelBufferRelease (page 152)
Releases a pixel buffer.

CVPixelBufferRetain (page 152)
Retains a pixel buffer.

CVPixelBufferUnlockBaseAddress (page 153)
Unlocks the base address of the pixel buffer.

CVPixelBufferPool Functions

CVPixelBufferPoolCreate (page 148)
Creates a pixel buffer pool.

CVPixelBufferPoolCreatePixelBuffer (page 149)
Creates a pixel buffer from a pixel buffer pool.

CVPixelBufferPoolGetAttributes (page 150)
Returns the pool attributes dictionary for a pixel buffer pool.

CVPixelBufferPoolGetPixelBufferAttributes (page 150)
Returns the attributes of pixel buffers that will be created from this pool.

CVPixelBufferPoolGetTypeID (page 151)
Returns the Core Foundation ID of the pixel buffer pool type.

CVPixelBufferPoolRelease (page 151)
Releases a pixel buffer pool.

CVPixelBufferPoolRetain (page 151)
Retains a pixel buffer pool.

CVPixelFormatDescription Functions

Used only if you are defining a custom pixel format.

CVPixelFormatDescriptionRegisterDescriptionWithPixelFormatType (page 154)
Registers a pixel format description with Core Video.

CVPixelFormatDescriptionCreateWithPixelFormatType (page 153)
Creates a pixel format description from a given OSType identifier.

CVPixelFormatDescriptionArrayCreateWithAllPixelFormatTypes (page 153)
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 163) 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 110) or `CVBufferSetAttachments` (page 111).

You can find predefined attachment keys in "CVBuffer Attachment Keys" (page 163) and "Image Buffer Attachment Keys" (page 167).

Availability

Available in Mac OS X v10.4 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 163) 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 163) and "Image Buffer Attachment Keys" (page 167).

Availability

Available in Mac OS X v10.4 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 v10.4 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 v10.4 and later.

Declared In
CVBuffer.h

CVBufferRemoveAllAttachments

Removes all attachments of a Core Video buffer.

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

Parameters

buffer

The Core Video buffer whose attachments you want to remove.

Discussion

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

Availability

Available in Mac OS X v10.4 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 163) and "Image Buffer Attachment Keys" (page 167).

Availability

Available in Mac OS X v10.4 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 v10.4 and later.

Declared In
`CVBuffer.h`

CVBufferSetAttachment
Sets or adds an attachment of a Core Video buffer.

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

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

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

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

attachmentMode
Specifies the attachment mode for this attachment. See "CVBuffer Attachment Modes" (page 163) 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 163) and "Image Buffer Attachment Keys" (page 167).

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 106) or `CVBufferGetAttachments` (page 107) functions.

Availability
Available in Mac OS X v10.4 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 110) 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 v10.4 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 177) 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 117).

Availability

Available in Mac OS X v10.4 and later.

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 177) for possible values.

Discussion

Use this call to create a display link for a single display.

Availability

Available in Mac OS X v10.4 and later.

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 177) 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 v10.4 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 177) for possible values.

Discussion
Using this function avoids having to call the Core Graphics function `CGOpenGLDisplayMaskToDisplayID`.

Availability
Available in Mac OS X v10.4 and later.

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 v10.4 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 v10.4 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 177) 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.

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 v10.4 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 v10.4 and later.

Declared In

`CVDisplayLink.h`

CVDisplayLinkGetTypeID

Obtains the Core Foundation ID for the display link data type.

```
CFTypeID CVDisplayLinkGetTypeID ();
```

Return Value

The Core Foundation ID for this type.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CVDisplayLink.h`

CVDisplayLinkIsRunning

Indicates whether a given display link is running.

```
Boolean CVDisplayLinkIsRunning (
    CVDisplayLink 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 v10.4 and later.

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 v10.4 and later.

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 v10.4 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 177) 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 v10.4 and later.

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 177) for possible values.

Discussion

This function chooses the display with the lowest refresh rate.

Availability

Available in Mac OS X v10.4 and later.

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 154) for more information about implementing this function.

userInfo

A pointer to user data.

Return Value

A Core Video result code. See "Result Codes" (page 177) for possible values.

Discussion

The display link invokes this callback whenever it wants you to output a frame.

Availability

Available in Mac OS X v10.4 and later.

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 177) 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 v10.4 and later.

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 177) 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 v10.4 and later.

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 177) for possible values.

Discussion
Note that the display link has to be running for this call to succeed.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CVDisplayLink.h`

CVGetCurrentHostTime

Retrieves the current value of the host time base.

```
uint64_t CVGetCurrentHostTime (void);
```

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 v10.4 and later.

Declared In
`CVHostTime.h`

CVGetHostClockFrequency

Retrieve the frequency of the host time base.

```
double CVGetHostClockFrequency (void);
```

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 v10.4 and later.

Declared In
`CVHostTime.h`

CVGetHostClockMinimumTimeDelta

Retrieve the smallest possible increment in the host time base.

```
uint32_t CVGetHostClockMinimumTimeDelta (void);
```

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

Availability
Available in Mac OS X v10.4 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 v10.4 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 v10.4 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 v10.4 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 v10.4 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 177) for possible values.

Availability
Available in Mac OS X v10.4 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 177) for possible values.

Discussion

Availability

Available in Mac OS X v10.4 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 v10.4 and later.

Declared In
`CVOpenGLBuffer.h`

CVOpenGLBufferGetTypeID
Obtains the Core Foundation type ID for the OpenGL buffer type.

```
CFTypeID CVOpenGLBufferGetTypeID();
```

Return Value
The Core Foundation ID for this data type.

Availability
Available in Mac OS X v10.4 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 177) for possible values.

Availability
Available in Mac OS X v10.4 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 177) for possible values.

Discussion
The function creates a new OpenGL buffer using the OpenGL buffer attributes specified in the CVOpenGLBufferPoolCreate (page 126) call. This buffer has default attachments as specified in the openGLBufferAttributes parameter of CVOpenGLBufferPoolCreate (page 126) (using either the kCVBufferPropagatedAttachmentsKey or kCVBufferNonPropagatedAttachmentsKey attributes).

Availability
Available in Mac OS X v10.4 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 v10.4 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 v10.4 and later.

Declared In
CVOpenGLBufferPool.h

CVOpenGLBufferPoolGetTypeID

Obtains the Core Foundation ID for the OpenGL buffer pool type.

```
CFTypeID CVOpenGLBufferPoolGetTypeID ();
```

Return Value
The Core Foundation ID for this data type.

Availability
Available in Mac OS X v10.4 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 v10.4 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 v10.4 and later.

Declared In

`CVOpenGLBufferPool.h`

CVOpenGLBufferRelease

Releases a Core Video OpenGL buffer.

```
void CVOpenGLBufferRelease (
    CVOpenGLBufferRef texture
);
```

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 v10.4 and later.

Declared In
`CVOpenGLBuffer.h`

CVOpenGLBufferRetain

Retains a Core Video OpenGL buffer.

```
CVOpenGLBufferRef CVOpenGLBufferRetain (
    CVOpenGLBufferRef texture
);
```

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 v10.4 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 177) for possible values.

Availability
Available in Mac OS X v10.4 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 177) 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 v10.4 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 177) 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 v10.4 and later.

Declared In
`CVOpenGLTextureCache.h`

CVOpenGLTextureCacheGetTypeID

Returns the Core Foundation ID of the texture cache type.

```
CFTypeID CVOpenGLTextureCacheGetTypeID ();
```

Return Value
The Core Foundation ID for this type.

Availability
Available in Mac OS X v10.4 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 v10.4 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 v10.4 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 v10.4 and later.

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 v10.4 and later.

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 v10.4 and later.

Declared In
`CVOpenGLTexture.h`

CVOpenGLTextureGetTypeID

Obtains the Core Foundation ID for the Core Video OpenGL texture type.

```
CFTypeID CVOpenGLTextureGetTypeID ();
```

Return Value
The Core Foundation ID for this type.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CVOpenGLBufferPool.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 v10.4 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 v10.4 and later.

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 v10.4 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 171) 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 177) 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 v10.4 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 177) 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 v10.4 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 156) 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 171) 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 177) 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 v10.4 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 156) 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 171) 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 177) 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 v10.4 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 v10.4 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 148).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CVPixelBuffer.h`

CVPixelBufferGetBaseAddressOfPlane

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

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

Parameters

pixelBuffer

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

planeIndex

The index of the plane.

Return Value

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

Discussion

Retrieving the base address for a pixel buffer requires that the buffer base address be locked by a successful call to `CVPixelBufferLockBaseAddress` (page 148).

Availability

Available in Mac OS X v10.4 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 v10.4 and later.

Declared In

`CVPixelBuffer.h`

CVPixelBufferGetBytesPerRowOfPlane

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

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

Parameters

pixelBuffer

The pixel buffer containing the plane.

planeIndex

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

Return Value

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

Availability

Available in Mac OS X v10.4 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 140).

Availability

Available in Mac OS X v10.4 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 v10.4 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 v10.4 and later.

Declared In

`CVPixelBuffer.h`

CVPixelBufferGetHeightOfPlane

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

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

Parameters

pixelBuffer

The pixel buffer whose plane height you want to obtain.

planeIndex

The index of the plane.

Return Value

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

Availability

Available in Mac OS X v10.4 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 v10.4 and later.

Declared In

`CVPixelBuffer.h`

CVPixelBufferGetPlaneCount

Returns number of planes of the pixel buffer.

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

Parameters

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

Return Value

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CVPixelBuffer.h`

CVPixelBufferGetTypeID

Returns the Core Foundation ID of the pixel buffer type.

```
CFTypeID CVPixelBufferGetTypeID ();
```

Return Value

The Core Foundation ID for this type.

Availability

Available in Mac OS X v10.4 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 v10.4 and later.

Declared In

`CVPixelBuffer.h`

CVPixelBufferGetWidthOfPlane

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

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

Parameters

pixelBuffer

The pixel buffer whose plane width you want to obtain.

planeIndex

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

Return Value

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

Availability

Available in Mac OS X v10.4 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 140), `false` otherwise.

Availability

Available in Mac OS X v10.4 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 177) for possible values.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CVPixelBuffer.h`

CVPixelBufferPoolCreate

Creates a pixel buffer pool.

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

Parameters

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

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

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

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

Return Value
A Core Video result code. See "Result Codes" (page 177) for possible values.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CVPixelBufferPool.h`

CVPixelBufferPoolCreatePixelBuffer

Creates a pixel buffer from a pixel buffer pool.

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

Parameters

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

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

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

Return Value
A Core Video result code. See "Result Codes" (page 177) 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 148) (using either the `kCVBufferPropagatedAttachmentsKey` or `kCVBufferNonPropagatedAttachmentsKey` attributes).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CVPixelBufferPool.h`

CVPixelBufferPoolGetAttributes
Returns the pool attributes dictionary for a pixel buffer pool.

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

Parameters
pool
The pixel buffer pool to retrieve the attributes from.

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

Discussion

Availability
Available in Mac OS X v10.4 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 v10.4 and later.

Declared In
`CVPixelBufferPool.h`

CVPixelBufferPoolGetTypeID

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

```
CFTypeID CVPixelBufferPoolGetTypeID ();
```

Return Value
The Core Foundation ID for this type.

Availability
Available in Mac OS X v10.4 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 v10.4 and later.

Declared In
`CVPixelBufferPool.h`

CVPixelBufferPoolRetain

Retains a pixel buffer pool.

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

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

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

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

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CVPixelBufferPool.h`

CVPixelBufferRelease

Releases a pixel buffer.

```
void CVPixelBufferRelease (
    CVPixelBufferRef buffer
);
```

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 v10.4 and later.

Declared In
`CVPixelBuffer.h`

CVPixelBufferRetain

Retains a pixel buffer.

```
CVPixelBufferRef CVPixelBufferRetain (
    CVPixelBufferRef buffer
);
```

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 v10.4 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 177) for possible values.

Discussion

Availability
Available in Mac OS X v10.4 and later.

Declared In
CVPixelBuffer.h

CVPixelFormatDescriptionArrayCreateWithAllPixelFormatTypes

Returns all the pixel format descriptions known to Core Video.

```
CFDictionaryRef 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 173) for a list of keys relevant to the format description.

Availability
Available in Mac OS X v10.3 and later.

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 173) for a list of keys relevant to the format description.

Availability

Available in Mac OS X v10.3 and later.

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 173) 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.

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 119) function when registering your callback.

Discussion

For a given display link, you must register a display link output callback using `CVDisplayLinkSetOutputCallback` (page 119) 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.

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 158) 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.

CVPixelBufferReleaseBytesCallback

Defines a pointer to a pixel buffer release callback function, which is called when a pixel buffer created by `CVPixelBufferCreateWithBytes` (page 138) 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 138).

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 138).

Discussion

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

CVPixelBufferReleasePlanarBytesCallback

Defines a pointer to a pixel buffer release callback function, which is called when a pixel buffer created by `CVPixelBufferCreateWithPlanarBytes` (page 140) 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 140).

dataPtr

A pointer to a plane descriptor block. This is the same pointer you passed to `CVPixelBufferCreateWithPlanarBytes` (page 140) in the `dataPtr` parameter.

dataSize

The size value you passed to `CVPixelBufferCreateWithPlanarBytes` (page 140) in the `dataSize` parameter.

numberOfPlanes

The number of planes value you passed to `CVPixelBufferCreateWithPlanarBytes` (page 140) in the `numberOfPlanes` parameter.

planeAddresses

A pointer to the base plane address you passed to `CVPixelBufferCreateWithPlanarBytes` (page 140) in the `basePlaneAddress` parameter.

Discussion

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

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.

CVDisplayLinkRef

Defines a display link.

```
typedef struct __CVDisplayLink *CVDisplayLinkRef;
```

Availability

Available in Mac OS X v10.3 and later.

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 141), 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 173).

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.

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.

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.

CVOpenGLBufferPoolRef

Defines an OpenGL buffer pool.

```
typedef struct _CVOpenGLBufferPool *CVOpenGLBufferPoolRef;
```

Availability
Available in Mac OS X v10.3 and later.

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.

CVOpenGLTextureCacheRef

Defines a CoreVideo OpenGL texture cache.

```
typedef struct __CVOpenGLTextureCache *CVOpenGLTextureCacheRef;
```

Availability
Available in Mac OS X v10.3 and later.

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.

CVPixelBufferPoolRef

Defines a pixel buffer pool.

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

Availability
Available in Mac OS X v10.3 and later.

CVReturn

Defines the return error code for Core Video functions.

```
typedef int32_t CVReturn;
```

Discussion
See "Result Codes" (page 177) for possible values.

Availability
Available in Mac OS X v10.3 and later.

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 176) for a list of possible values.

`flags`
A set of flags that indicate the SMPTE state. See "SMPTE State Flags" (page 176) 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.

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 164) 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.

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 165) 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.

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.

`kCVBufferTimeValueKey`
The actual time value associated with the movie.

Available in Mac OS X v10.3 and later.

`kCVBufferTimeScaleKey`
The time scale associated with the movie.

Available in Mac OS X v10.3 and later.

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.

`kCVAttachmentMode_ShouldPropagate`
Copy this attachment when using the `CVBufferPropagateAttachments` (page 108) 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.

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 108) function (type `CFDictionary`, containing a list of attachments as key-value pairs).

Available in Mac OS X v10.3 and later.

`kCVBufferNonPropagatedAttachmentsKey`
Attachments that should not be copied when using the `CVBufferPropagateAttachments` (page 108) function (type `CFDictionary`, containing a list of attachments as key-value pairs).

Available in Mac OS X v10.3 and later.

Discussion

These attributes let you set multiple attachments at the time of buffer creation, rather than having to call `CVBufferSetAttachment` (page 110) 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.

CVTime Values

Indicate specific `CVTime` values.

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

Constants

`kCVZeroTime`
Zero time or duration. For example, `CVDisplayLinkGetOutputVideoLatency` (page 115) returns `kCVZeroTime` for zero video latency.

Available in Mac OS X v10.3 and later.

`kCVIndefiniteTime`
An unknown or indefinite time. For example, `CVDisplayLinkGetNominalOutputVideoRefreshPeriod` (page 115) returns `kCVIndefiniteTime` if the display link specified is not valid.

Available in Mac OS X v10.3 and later.

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.

`kCVTimeStampHostTimeValid`
The value in the host time field is valid.

Available in Mac OS X v10.3 and later.

`kCVTimeStampSMPTETimeValid`
The value in the SMPTE time field is valid.

Available in Mac OS X v10.3 and later.

`kCVTimeStampVideoRefreshPeriodValid`
The value in the video refresh period field is valid.

Available in Mac OS X v10.3 and later.

`kCVTimeStampRateScalarValid`
The value in the rate scalar field is valid.

Available in Mac OS X v10.3 and later.

`kCVTimeStampTopField`
The timestamp represents the top lines of an interlaced image.

Available in Mac OS X v10.3 and later.

`kCVTimeStampBottomField`
The timestamp represents the bottom lines of an interlaced image.

Available in Mac OS X v10.3 and later.

`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.

`kCVTimeStampIsInterlaced`
A convenience constant indicating that the timestamp is for an interlaced image.

Available in Mac OS X v10.3 and later.

Discussion
These flags indicate which fields in the `CVTimeStamp` (page 162) 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.

`kCVImageBufferGammaLevelKey`
The gamma level for this buffer (type `CFNumber`).

Available in Mac OS X v10.3 and later.

`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.

`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.

`kCVImageBufferCleanApertureWidthKey`
The clean aperture width (type `CFNumber`).

Available in Mac OS X v10.3 and later.

`kCVImageBufferCleanApertureHeightKey`
The clean aperture height (type `CFNumber`).

Available in Mac OS X v10.3 and later.

`kCVImageBufferCleanApertureHorizontalOffsetKey`
The clean aperture horizontal offset (type `CFNumber`).

Available in Mac OS X v10.3 and later.

`kCVImageBufferCleanApertureVerticalOffsetKey`
The clean aperture vertical offset (type `CFNumber`).

Available in Mac OS X v10.3 and later.

`kCVImageBufferFieldCountKey`
The field count for the buffer (type `CFNumber`).

Available in Mac OS X v10.3 and later.

`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.

`kCVImageBufferFieldDetailTemporalTopFirst`
(type `CFString`).

Available in Mac OS X v10.3 and later.

`kCVImageBufferFieldDetailTemporalBottomFirst`
(type `CFString`).

Available in Mac OS X v10.3 and later.

`kCVImageBufferFieldDetailSpatialFirstLineEarly`
(type `CFString`).

Available in Mac OS X v10.3 and later.

`kCVImageBufferFieldDetailSpatialFirstLineLate`
(type `CFString`).

Available in Mac OS X v10.3 and later.

`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.

`kCVImageBufferPixelAspectRatioHorizontalSpacingKey`
The horizontal component of the buffer aspect ratio (type `CFNumber`).

Available in Mac OS X v10.3 and later.

`kCVImageBufferPixelAspectRatioVerticalSpacingKey`
The vertical component of the buffer aspect ratio (type `CFNumber`).

Available in Mac OS X v10.3 and later.

`kCVImageBufferDisplayDimensionsKey`
The buffer display dimensions (type `CFDictionary` containing the buffer display width and height keys).

Available in Mac OS X v10.3 and later.

`kCVImageBufferDisplayWidthKey`
The buffer display width (type `CFNumber`).

Available in Mac OS X v10.3 and later.

`kCVImageBufferDisplayHeightKey`
The buffer display height (type `CFNumber`).

Available in Mac OS X v10.3 and later.

`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.

`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.

`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.

`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.

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.

`kCVOpenGLBufferHeight`
The height of the buffer.

Available in Mac OS X v10.3 and later.

`kCVOpenGLBufferTarget`
The OpenGL target for this buffer.

Available in Mac OS X v10.3 and later.

`kCVOpenGLBufferInternalFormat`
The OpenGL internal format of this buffer.

Available in Mac OS X v10.3 and later.

`kCVOpenGLBufferMaximumMipmapLevel`
The maximum mipmap level for this buffer.

Available in Mac OS X v10.3 and later.

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.

`kCVOpenGLBufferPoolMaximumBufferAgeKey`
Indicates how long unused buffers should be kept before they are deallocated (type `CFAbsoluteTime`).

Available in Mac OS X v10.3 and later.

Discussion
You specify these keys in a Core Foundation dictionary when calling functions such as `CVOpenGLBufferPoolCreate` (page 126).

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.

`kCVPixelBufferMemoryAllocatorKey`
The allocator used with this buffer (type `CFAllocatorRef`).

Available in Mac OS X v10.3 and later.

`kCVPixelBufferWidthKey`
The width of the pixel buffer (type `CFNumber`).

Available in Mac OS X v10.3 and later.

`kCVPixelBufferHeightKey`
The height of the pixel buffer (type `CFNumber`).

Available in Mac OS X v10.3 and later.

`kCVPixelBufferExtendedPixelsLeftKey`
The number of pixels padding the left of the image (type `CFNumber`).

Available in Mac OS X v10.3 and later.

`kCVPixelBufferExtendedPixelsTopKey`
The number of pixels padding the top of the image (type `CFNumber`).

Available in Mac OS X v10.3 and later.

`kCVPixelBufferExtendedPixelsRightKey`
The number of pixels padding the right of the image (type `CFNumber`).

Available in Mac OS X v10.3 and later.

`kCVPixelBufferExtendedPixelsBottomKey`
The number of pixels padding the bottom of the image (type `CFNumber`).

Available in Mac OS X v10.3 and later.

`kCVPixelBufferBytesPerRowAlignmentKey`
Indicates the number of bytes per row in the pixel buffer (type `CFNumber`).

Available in Mac OS X v10.3 and later.

`kCVPixelBufferCGBitmapContextCompatibilityKey`
Indicates whether the pixel buffer is compatible with Core Graphics bitmap contexts (type `CFBoolean`).

Available in Mac OS X v10.3 and later.

`kCVPixelBufferCGImageCompatibilityKey`
Indicates whether the pixel buffer is compatible with `CGImage` types (type `CFBoolean`).

Available in Mac OS X v10.3 and later.

`kCVPixelBufferOpenGLCompatibilityKey`
Indicates whether the pixel buffer is compatible with OpenGL contexts (type `CFBoolean`).

Available in Mac OS X v10.3 and later.

Discussion
You specify these keys in a Core Foundation dictionary when calling functions such as `CVPixelBufferCreate` (page 137).

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.

`kCVPixelBufferPoolMaximumBufferAgeKey`
The maximum allowable age for a buffer in the pixel buffer pool (type `CFAbsoluteTime`).

Available in Mac OS X v10.3 and later.

Discussion

You specify these keys in a Core Foundation dictionary when calling functions such as `CVPixelBufferPoolCreate` (page 148).

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.

`kCVPixelFormatConstant`

The pixel format constant for QuickTime.

Available in Mac OS X v10.3 and later.

`kCVPixelFormatCodecType`

The codec type (type `CFString`). For example, `'2vuy'` or `k422YpCbCr8CodecType`.

Available in Mac OS X v10.3 and later.

`kCVPixelFormatFourCC`

The Microsoft FourCC equivalent code for this pixel format (type `CFString`).

Available in Mac OS X v10.3 and later.

`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.

`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.

`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.

`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.

`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.

`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.

`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.

`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.

`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.

`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.

`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.

`kCVPixelFormatCGBitmapInfo`
The Core Graphics bitmap information for this pixel format (if applicable).

Available in Mac OS X v10.3 and later.

`kCVPixelFormatQDCompatibility`
Indicates whether this format is compatible with QuickDraw (type `CFBoolean`).

Available in Mac OS X v10.3 and later.

`kCVPixelFormatCGBitmapContextCompatibility`
Indicates whether this format is compatible with Core Graphics bitmap contexts(type `CFBoolean`).

Available in Mac OS X v10.3 and later.

`kCVPixelFormatCGImageCompatibility`
Indicates whether this format is compatible with the `CGImage` type (type `CFBoolean`).

Available in Mac OS X v10.3 and later.

`kCVPixelFormatOpenGLCompatibility`
Indicates whether this format is compatible with OpenGL (type `CFBoolean`).

Available in Mac OS X v10.3 and later.

`kCVPixelFormatFillExtendedPixelsCallback`
Specifies a custom extended pixel fill algorithm (type `CFData`). See `CVFillExtendedPixelsCallBack` (page 155) and `CVFillExtendedPixelsCallbackData` (page 158) for more information.

Available in Mac OS X v10.3 and later.

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.

`kCVSMPTETimeRunning`
Time is running.

Available in Mac OS X v10.3 and later.

Discussion
You use these values in the `CVSMPTETime` (page 161) 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.

`kCVSMPTETimeType25`
25 frames per second (standard PAL).

Available in Mac OS X v10.3 and later.

`kCVSMPTETimeType30Drop`
30 drop frame.

Available in Mac OS X v10.3 and later.

`kCVSMPTETimeType30`
30 frames per second.

Available in Mac OS X v10.3 and later.

`kCVSMPTETimeType2997`
29.97 frames per second (standard NTSC).

Available in Mac OS X v10.3 and later.

`kCVSMPTETimeType2997Drop`
29.97 drop frame.

Available in Mac OS X v10.3 and later.

`kCVSMPTETimeType60`
60 frames per second.

Available in Mac OS X v10.3 and later.

`kCVSMPTETimeType5994`
59.94 frames per second.

Available in Mac OS X v10.3 and later.

Discussion
You use these values in the `CVSMPTETime` (page 161) 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
`kCVReturnFirst`	-6660	Placeholder to mark the beginning of Core Video result codes (not returned by any functions).
`kCVReturnError`	-6660	An otherwise undefined error occurred.
`kCVReturnInvalidArgument`	-6661	Invalid function parameter. For example, out of range or the wrong type.
`kCVReturnAllocationFailed`	-6662	Memory allocation for a buffer or buffer pool failed.

Result Code	Value	Description
kCVReturnInvalidDisplay	-6670	The display specified when creating a display link is invalid.
kCVReturnDisplayLinkAlreadyRunning	-6671	The specified display link is already running.
kCVReturnDisplayLinkNotRunning	-6672	The specified display link is not running.
kCVReturnDisplayLinkCallbacksNotSet	-6673	No callback registered for the specified display link. You must set either the output callback or both the render and display callbacks.
kCVReturnInvalidPixelFormat	-6600	The buffer does not support the specified pixel format.
kCVReturnInvalidSize	-6601	The buffer cannot support the requested buffer size (usually too big).
kCVReturnInvalidPixelBufferAttributes	-6602	A buffer cannot be created with the specified attributes.
kCVReturnPixelBufferNotOpenGLCompatible	-6603	The pixel buffer is not compatible with OpenGL due to an unsupported buffer size, pixel format, or attribute.
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.
kCVReturnInvalidPoolAttributes	-6691	A buffer pool cannot be created with the specified attributes.
kCVReturnLast	-6699	Placeholder to mark the end of Core Video result codes (not returned by any functions).

Document Revision History

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

Date	Notes
2006-05-23	First publication of this content as a collection of separate documents.

Document Revision History

Index

D

E

F

G

I

K

L

N

O

P

R

S

T

U

V

W

X

Y

Z