CGImage Reference

Graphics & Animation: 2D Drawing

2010-08-03

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

Simultaneously published in the United States and Canada.

Contents

CGImage Reference

Derived From:	*CFType Reference*
Framework:	ApplicationServices/ApplicationServices.h
Companion guide	Quartz 2D Programming Guide
Declared in	CGImage.h

Overview

The `CGImageRef` opaque type represents bitmap images and bitmap image masks, based on sample data that you supply. A bitmap (or sampled) image is a rectangular array of pixels, with each pixel representing a single sample or data point in a source image.

Functions by Task

Creating Bitmap Images

`CGImageCreate` (page 7)
Creates a bitmap image from data supplied by a data provider.

`CGImageCreateCopy` (page 8)
Creates a copy of a bitmap image.

`CGImageCreateCopyWithColorSpace` (page 9)
Create a copy of a bitmap image, replacing its colorspace.

`CGImageCreateWithJPEGDataProvider` (page 10)
Creates a bitmap image using JPEG-encoded data supplied by a data provider.

`CGImageCreateWithPNGDataProvider` (page 12)
Creates a Quartz bitmap image using PNG-encoded data supplied by a data provider.

`CGImageCreateWithImageInRect` (page 9)
Creates a bitmap image using the data contained within a subregion of an existing bitmap image.

`CGImageCreateWithMask` (page 11)
Creates a bitmap image from an existing image and an image mask.

`CGImageCreateWithMaskingColors` (page 11)
Creates a bitmap image by masking an existing bitmap image with the provided color values.

Creating an Image Mask

`CGImageMaskCreate` (page 18)
Creates a bitmap image mask from data supplied by a data provider.

Retaining and Releasing Images

`CGImageRetain` (page 20)
Increments the retain count of a bitmap image.

`CGImageRelease` (page 20)
Decrements the retain count of a bitmap image.

Getting the CFType ID

`CGImageGetTypeID` (page 17)
Returns the type identifier for Quartz bitmap images.

Getting Information About an Image

`CGImageGetAlphaInfo` (page 13)
Returns the alpha channel information for a bitmap image.

`CGImageGetBitmapInfo` (page 13)
Returns the bitmap information for a bitmap image.

`CGImageGetBitsPerComponent` (page 14)
Returns the number of bits allocated for a single color component of a bitmap image.

`CGImageGetBitsPerPixel` (page 14)
Returns the number of bits allocated for a single pixel in a bitmap image.

`CGImageGetBytesPerRow` (page 14)
Returns the number of bytes allocated for a single row of a bitmap image.

`CGImageGetColorSpace` (page 15)
Return the color space for a bitmap image.

`CGImageGetDataProvider` (page 15)
Returns the data provider for a bitmap image.

`CGImageGetDecode` (page 15)
Returns the decode array for a bitmap image.

`CGImageGetHeight` (page 16)
Returns the height of a bitmap image.

`CGImageGetShouldInterpolate` (page 17)
Returns the interpolation setting for a bitmap image.

`CGImageGetRenderingIntent` (page 16)
Returns the rendering intent setting for a bitmap image.

`CGImageGetWidth` (page 18)
Returns the width of a bitmap image.

`CGImageIsMask` (page 18)
Returns whether a bitmap image is an image mask.

Functions

CGImageCreate

Creates a bitmap image from data supplied by a data provider.

```
CGImageRef CGImageCreate (
   size_t width,
   size_t height,
   size_t bitsPerComponent,
   size_t bitsPerPixel,
   size_t bytesPerRow,
   CGColorSpaceRef colorspace,
   CGBitmapInfo bitmapInfo,
   CGDataProviderRef provider,
   const CGFloat decode[],
   bool shouldInterpolate,
   CGColorRenderingIntent intent
);
```

Parameters

width
The width, in pixels, of the required image.

height
The height, in pixels, of the required image

bitsPerComponent
The number of bits for each component in a source pixel. For example, if the source image uses the RGBA-32 format, you would specify 8 bits per component.

bitsPerPixel
The total number of bits in a source pixel. This value must be at least `bitsPerComponent` times the number of components per pixel.

bytesPerRow
The number of bytes of memory for each horizontal row of the bitmap.

colorspace
The color space for the image. Quartz retains the color space you pass in; on return, you may safely release it.

bitmapInfo
A `CGBitmapInfo` constant that specifies whether the bitmap should contain an alpha channel and its relative location in a pixel, along with whether the components are floating-point or integer values.

provider
The source of data for the bitmap. For information about supported data formats, see the discussion below. Quartz retains this object; on return, you may safely release it.

decode

The decode array for the image. If you do not want to allow remapping of the image's color values, pass `NULL` for the decode array. For each color component in the image's color space (including the alpha component), a decode array provides a pair of values denoting the upper and lower limits of a range. For example, the decode array for a source image in the RGB color space would contain six entries total, consisting of one pair each for red, green, and blue. When the image is rendered, Quartz uses a linear transform to map the original component value into a relative number within your designated range that is appropriate for the destination color space.

shouldInterpolate

A Boolean value that specifies whether interpolation should occur. The interpolation setting specifies whether Quartz should apply a pixel-smoothing algorithm to the image. Without interpolation, the image may appear jagged or pixelated when drawn on an output device with higher resolution than the image data.

intent

A rendering intent constant that specifies how Quartz should handle colors that are not located within the gamut of the destination color space of a graphics context. The rendering intent determines the exact method used to map colors from one color space to another. For descriptions of the defined rendering-intent constants, see `Color Rendering Intents`.

Return Value

A new Quartz bitmap image. You are responsible for releasing this object by calling `CGImageRelease` (page 20).

Discussion

The data provider should provide raw data that matches the format specified by the other input parameters. To use encoded data (for example, from a file specified by a URL-based data provider), see `CGImageCreateWithJPEGDataProvider` (page 10) and `CGImageCreateWithPNGDataProvider` (page 12). In Mac OS X version 10.3 and later, you can also use the QuickTime function `GraphicsImportCreateCGImage` to decode an image file in any supported format and create a CGImage, in a single operation.

For information on supported pixel formats, see *Quartz 2D Programming Guide*.

Availability

Declared In

`CGImage.h`

CGImageCreateCopy

Creates a copy of a bitmap image.

```
CGImageRef CGImageCreateCopy (
   CGImageRef image
);
```

Parameters

image

The image to copy.

Return Value

An copy of the image specified by the `image` parameter.

Availability

Declared In
`CGImage.h`

CGImageCreateCopyWithColorSpace

Create a copy of a bitmap image, replacing its colorspace.

```
CGImageRef CGImageCreateCopyWithColorSpace (
   CGImageRef image,
   CGColorSpaceRef colorspace
);
```

Parameters

image

The graphics image to copy.

colorspace

The destination color space. The number of components in this color space must be the same as the number in the specified image.

Return Value
A new Quartz image that is a copy of the image passed as the `image` parameter but with its color space replaced by that specified by the `colorspace` parameter. Returns `NULL` if `image` is an image mask, or if the number of components of `colorspace` is not the same as the number of components of the colorspace of `image`. You are responsible for releasing this object using `CGImageRelease` (page 20).

Availability

Declared In
`CGImage.h`

CGImageCreateWithImageInRect

Creates a bitmap image using the data contained within a subregion of an existing bitmap image.

```
CGImageRef CGImageCreateWithImageInRect (
   CGImageRef image,
   CGRect rect
);
```

Parameters

image

The image to extract the subimage from.

rect

A rectangle whose coordinates specify the area to create an image from.

Return Value
A CGImage object that specifies a subimage of the image. If the `rect` parameter defines an area that is not in the image, returns `NULL`.

Discussion
Quartz performs these tasks to create the subimage:

- Adjusts the area specified by the `rect` parameter to integral bounds by calling the function `CGRectIntegral`.
- Intersects the result with a rectangle whose origin is (`0`,`0`) and size is equal to the size of the image specified by the `image` parameter.
- References the pixels within the resulting rectangle, treating the first pixel within the rectangle as the origin of the subimage.

If `W` and `H` are the width and height of image, respectively, then the point (`0`,`0`) corresponds to the first pixel of the image data. The point (`W-1`, `0`) is the last pixel of the first row of the image data while (`0`, `H-1`) is the first pixel of the last row of the image data and (`W-1`, `H-1`) is the last pixel of the last row of the image data.

The resulting image retains a reference to the original image, which means you may release the original image after calling this function.

Availability

Declared In
`CGImage.h`

CGImageCreateWithJPEGDataProvider

Creates a bitmap image using JPEG-encoded data supplied by a data provider.

```
CGImageRef CGImageCreateWithJPEGDataProvider (
   CGDataProviderRef source,
   const CGFloat decode[],
   bool shouldInterpolate,
   CGColorRenderingIntent intent
);
```

Parameters

source
: A data provider supplying JPEG-encoded data.

decode
: The decode array for the image. Typically a decode array is unnecessary, and you should pass `NULL`.

shouldInterpolate
: A Boolean value that specifies whether interpolation should occur. The interpolation setting specifies whether Quartz should apply a pixel-smoothing algorithm to the image.

intent
: A `CGColorRenderingIntent` constant that specifies how Quartz should handle colors that are not located within the gamut of the destination color space of a graphics context.

Return Value
A new Quartz bitmap image. You are responsible for releasing this object by calling `CGImageRelease` (page 20).

Availability

Declared In
`CGImage.h`

CGImageCreateWithMask

Creates a bitmap image from an existing image and an image mask.

```
CGImageRef CGImageCreateWithMask (
   CGImageRef image,
   CGImageRef mask
);
```

Parameters

image

The image to apply the `mask` parameter to. This image must not be an image mask and may not have an image mask or masking color associated with it.

mask

A mask. If the mask is an image, it must be in the DeviceGray color space, must not have an alpha component, and may not itself be masked by an image mask or a masking color. If the mask is not the same size as the image specified by the `image` parameter, then Quartz scales the mask to fit the image.

Return Value

An image created by masking `image` with `mask`. You are responsible for releasing this object by calling `CGImageRelease` (page 20).

Discussion

The resulting image depends on whether the `mask` parameter is an image mask or an image. If the `mask` parameter is an image mask, then the source samples of the image mask act as an inverse alpha value. That is, if the value of a source sample in the image mask is S, then the corresponding region in `image` is blended with the destination using an alpha value of (1-S). For example, if S is 1, then the region is not painted, while if S is 0, the region is fully painted.

If the `mask` parameter is an image, then it serves as an alpha mask for blending the image onto the destination. The source samples of `mask`' act as an alpha value. If the value of the source sample in mask is S, then the corresponding region in image is blended with the destination with an alpha of S. For example, if S is 0, then the region is not painted, while if S is 1, the region is fully painted.

Availability

Declared In

`CGImage.h`

CGImageCreateWithMaskingColors

Creates a bitmap image by masking an existing bitmap image with the provided color values.

```
CGImageRef CGImageCreateWithMaskingColors (
   CGImageRef image,
   const CGFloat components[]
);
```

Parameters

image

The image to mask. This parameter may not be an image mask, may not already have an image mask or masking color associated with it, and cannot have an alpha component.

components

An array of color components that specify a color or range of colors to mask the image with. The array must contain 2N values { min[1], max[1], ... min[N], max[N] } where N is the number of components in color space of `image`. Each value in `components` must be a valid image sample value. If `image` has integer pixel components, then each value must be in the range [0 .. 2**`bitsPerComponent` - 1] (where `bitsPerComponent` is the number of bits/component of `image`). If `image` has floating-point pixel components, then each value may be any floating-point number which is a valid color component.

Return Value

An image created by masking `image` with the colors specified in the `components` array. You are responsible for releasing this object by calling `CGImageRelease` (page 20).

Discussion

Any image sample with color value {c[1], ... c[N]} where min[i] <= c[i] <= max[i] for 1 <= i <= N is masked out (that is, not painted). This means that anything underneath the unpainted samples, such as the current fill color, shows through.

Availability

Declared In

`CGImage.h`

CGImageCreateWithPNGDataProvider

Creates a Quartz bitmap image using PNG-encoded data supplied by a data provider.

```
CGImageRef CGImageCreateWithPNGDataProvider (
   CGDataProviderRef source,
   const CGFloat decode[],
   bool shouldInterpolate,
   CGColorRenderingIntent intent
);
```

Parameters

source

A data provider supplying PNG-encoded data.

decode

The decode array for the image. Typically a decode array is unnecessary, and you should pass `NULL`.

shouldInterpolate

A Boolean value that specifies whether interpolation should occur. The interpolation setting specifies whether Quartz should apply a pixel-smoothing algorithm to the image.

intent

A `CGColorRenderingIntent` constant that specifies how Quartz should handle colors that are not located within the gamut of the destination color space of a graphics context.

Return Value

A new Quartz bitmap image. You are responsible for releasing this object by calling `CGImageRelease` (page 20).

Availability

Declared In

`CGImage.h`

CGImageGetAlphaInfo

Returns the alpha channel information for a bitmap image.

```
CGImageAlphaInfo CGImageGetAlphaInfo (
   CGImageRef image
);
```

Parameters

image

The image to examine.

Return Value

A `CGImageAlphaInfo` constant that specifies (1) whether the bitmap contains an alpha channel, (2) where the alpha bits are located in the image data, and (3) whether the alpha value is premultiplied. For possible values, see "Constants" (page 21). The function returns `kCGImageAlphaNone` if the `image` parameter refers to an image mask.

Discussion

The alpha value is what determines the opacity of a pixel when it is drawn.

Availability

Declared In

`CGImage.h`

CGImageGetBitmapInfo

Returns the bitmap information for a bitmap image.

```
CGBitmapInfo CGImageGetBitmapInfo (
   CGImageRef image
);
```

Parameters

image

An image.

Return Value

The bitmap information associated with an image.

Discussion

This function returns a constant that specifies:

- The type of bitmap data—floating point or integer. You use the constant `kCGBitmapFloatComponents` to extract this information.
- Whether an alpha channel is in the data, and if so, how the alpha data is stored. You use the constant `kCGBitmapAlphaInfoMask` to extract the alpha information. Alpha information is specified as one of the constants listed in "Alpha Information for Images" (page 21).

You can extract the alpha information

Availability

Declared In

`CGImage.h`

CGImageGetBitsPerComponent

Returns the number of bits allocated for a single color component of a bitmap image.

```
size_t CGImageGetBitsPerComponent (
   CGImageRef image
);
```

Parameters

image

The image to examine.

Return Value

The number of bits used in memory for each color component of the specified bitmap image (or image mask). Possible values are 1, 2, 4, or 8. For example, for a 16-bit RGB(A) colorspace, the function would return a value of 4 bits per color component.

Availability

Declared In

CGImage.h

CGImageGetBitsPerPixel

Returns the number of bits allocated for a single pixel in a bitmap image.

```
size_t CGImageGetBitsPerPixel (
   CGImageRef image
);
```

Parameters

image

The image to examine.

Return Value

The number of bits used in memory for each pixel of the specified bitmap image (or image mask).

Availability

Declared In

CGImage.h

CGImageGetBytesPerRow

Returns the number of bytes allocated for a single row of a bitmap image.

```
size_t CGImageGetBytesPerRow (
   CGImageRef image
);
```

Parameters

image

The image to examine.

Return Value

The number of bytes used in memory for each row of the specified bitmap image (or image mask).

Availability

Declared In
`CGImage.h`

CGImageGetColorSpace

Return the color space for a bitmap image.

```
CGColorSpaceRef CGImageGetColorSpace (
   CGImageRef image
);
```

Parameters

image
The image to examine.

Return Value
The source color space for the specified bitmap image, or `NULL` if the image is an image mask. You are responsible for retaining and releasing the color space as necessary.

Availability

Related Sample Code
aurioTouch
GLSprite

Declared In
`CGImage.h`

CGImageGetDataProvider

Returns the data provider for a bitmap image.

```
CGDataProviderRef CGImageGetDataProvider (
   CGImageRef image
);
```

Parameters

image
The image to examine.

Return Value
The data provider for the specified bitmap image (or image mask). You are responsible for retaining and releasing the data provider as necessary.

Availability

Declared In
`CGImage.h`

CGImageGetDecode

Returns the decode array for a bitmap image.

```
const CGFloat * CGImageGetDecode (
   CGImageRef image
);
```

Parameters

image

The image to examine.

Return Value

The decode array for a bitmap image (or image mask). See the discussion for a description of possible return values.

Discussion

For a bitmap image or image mask, for each color component in the source color space, the decode array contains a pair of values denoting the upper and lower limits of a range. When the image is rendered, Quartz uses a linear transform to map the original component value into a relative number, within the designated range, that is appropriate for the destination color space. If remapping of the image's color values is not allowed, the returned value will be `NULL`.

Availability

Declared In

`CGImage.h`

CGImageGetHeight

Returns the height of a bitmap image.

```
size_t CGImageGetHeight (
   CGImageRef image
);
```

Parameters

image

The image to examine.

Return Value

The height in pixels of the bitmap image (or image mask).

Availability

Related Sample Code

aurioTouch

GLSprite

Declared In

`CGImage.h`

CGImageGetRenderingIntent

Returns the rendering intent setting for a bitmap image.

```
CGColorRenderingIntent CGImageGetRenderingIntent (
   CGImageRef image
);
```

Parameters

image

The image to examine.

Return Value

Returns the `CGColorRenderingIntent` constant that specifies how Quartz should handle colors that are not located within the gamut of the destination color space of a graphics context in which the image is drawn. If the image is an image mask, this function returns `kCGRenderingIntentDefault`.

Availability

Declared In

`CGImage.h`

CGImageGetShouldInterpolate

Returns the interpolation setting for a bitmap image.

```
bool CGImageGetShouldInterpolate (
   CGImageRef image
);
```

Parameters

image

The image to examine.

Return Value

Returns `1` if interpolation is enabled for the specified bitmap image (or image mask), otherwise, returns `0`.

Discussion

The interpolation setting specifies whether Quartz should apply an edge-smoothing algorithm to the associated image.

Availability

Declared In

`CGImage.h`

CGImageGetTypeID

Returns the type identifier for Quartz bitmap images.

```
CFTypeID CGImageGetTypeID (
   void
);
```

Return Value

The identifier for the opaque type `CGImageRef` (page 21).

Availability

Declared In
`CGImage.h`

CGImageGetWidth

Returns the width of a bitmap image.

```
size_t CGImageGetWidth (
   CGImageRef image
);
```

Parameters

image
The image to examine.

Return Value
The width, in pixels, of the specified bitmap image (or image mask).

Availability

Related Sample Code
aurioTouch
GLSprite

Declared In
`CGImage.h`

CGImageIsMask

Returns whether a bitmap image is an image mask.

```
bool CGImageIsMask (
   CGImageRef image
);
```

Parameters

image
The image to examine.

Return Value
A Boolean value that indicates whether the image passed in the `image` parameter is an image mask (`true` indicates that the image is an image mask).

Availability

Declared In
`CGImage.h`

CGImageMaskCreate

Creates a bitmap image mask from data supplied by a data provider.

```
CGImageRef CGImageMaskCreate (
   size_t width,
   size_t height,
   size_t bitsPerComponent,
   size_t bitsPerPixel,
   size_t bytesPerRow,
   CGDataProviderRef provider,
   const CGFloat decode[],
   bool shouldInterpolate
);
```

Parameters

width

The width, in pixels, of the required image mask.

height

The height, in pixels, of the required image mask.

bitsPerComponent

The number of significant masking bits in a source pixel. For example, if the source image is an 8-bit mask, you specify 8 bits per component. Image masks must be 1, 2, 4, or 8 bits per component.

bitsPerPixel

The total number of bits in a source pixel.

bytesPerRow

The number of bytes to use for each horizontal row of the image mask.

provider

The data source for the image mask.

decode

Typically a decode array is unnecessary, and you should pass `NULL`.

shouldInterpolate

A Boolean value that specifies whether interpolation should occur. The interpolation setting specifies whether Quartz should apply an edge-smoothing algorithm to the image mask.

Return Value

A Quartz bitmap image mask. You are responsible for releasing this object by calling `CGImageRelease` (page 20).

Discussion

A Quartz bitmap image mask is used the same way an artist uses a silkscreen, or a sign painter uses a stencil. The bitmap represents a mask through which a color is transferred. The bitmap itself does not have a color. It gets its color from the fill color currently set in the graphics state.

When you draw into a context with a bitmap image mask, Quartz uses the mask to determine where and how the current fill color is applied to the image rectangle. Each sample value in the mask specifies how much of the current fill color is masked out at a specific location. Effectively, the sample value specifies the opacity of the mask. Larger values represent greater opacity and hence less color applied to the page.

Image masks must be 1, 2, 4, or 8 bits per component. For a 1-bit mask, a sample value of `1` specifies sections of the mask that are masked out; these sections block the current fill color. A sample value of `0` specifies sections of the mask that are not masked out; these sections show the current fill color of the graphics state when the mask is painted. You can think of the sample values as an inverse alpha. That is, a value of `1` is transparent and `0` is opaque.

For image masks that are 2, 4, or 8 bits per component, each component is mapped to a range of 0 to 1 by scaling using this formula:

```
1/(2^bits per component - 1)
```

For example, a 4-bit mask has values that range from 0 to 15. These values are scaled by 1/15 so that each component ranges from 0 to 1. Component values that rescale to 0 or 1 behave the same way as they behave for 1-bit image masks. Values that scale to between 0 and 1 act as an inverse alpha. That is, the fill color is painted as if it has an alpha value of (`1 - MaskSampleValue`). For example, if the sample value of an 8-bit mask scales to `0.8`, the current fill color is painted as if it has an alpha value of `0.2`, that is (`1-0.8`).

Availability

Declared In
`CGImage.h`

CGImageRelease

Decrements the retain count of a bitmap image.

```
void CGImageRelease (
   CGImageRef image
);
```

Parameters

image
The image to release.

Discussion
This function is equivalent to `CFRelease`, except that it does not cause an error if the `image` parameter is `NULL`.

Availability

Related Sample Code
aurioTouch

Declared In
`CGImage.h`

CGImageRetain

Increments the retain count of a bitmap image.

```
CGImageRef CGImageRetain (
   CGImageRef image
);
```

Parameters

image
The image to retain.

Return Value
The same image you passed in as the `image` parameter.

Discussion
This function is equivalent to `CFRetain`, except that it does not cause an error if the `image` parameter is `NULL`.

Availability

Declared In
`CGImage.h`

Data Types

CGImageRef

An opaque type that encapsulates bitmap image information.

```
typedef struct CGImage *CGImageRef;
```

Availability
Available in iOS 2.0 and later.

Declared In
`CGImage.h`

Constants

Alpha Information for Images

Storage options for alpha component data.

```
enum CGImageAlphaInfo {
    kCGImageAlphaNone,
    kCGImageAlphaPremultipliedLast,
    kCGImageAlphaPremultipliedFirst,
    kCGImageAlphaLast,
    kCGImageAlphaFirst,
    kCGImageAlphaNoneSkipLast,
    kCGImageAlphaNoneSkipFirst
};
typedef enum CGImageAlphaInfo CGImageAlphaInfo;
```

Constants
`kCGImageAlphaFirst`

The alpha component is stored in the most significant bits of each pixel. For example, non-premultiplied ARGB.

Available in iOS 2.0 and later.

Declared in `CGImage.h`.

`kCGImageAlphaLast`
The alpha component is stored in the least significant bits of each pixel. For example, non-premultiplied RGBA.

Available in iOS 2.0 and later.

Declared in `CGImage.h`.

`kCGImageAlphaNone`
There is no alpha channel. If the total size of the pixel is greater than the space required for the number of color components in the color space, the least significant bits are ignored. This value is equivalent to `kCGImageAlphaNoneSkipLast`.

Available in iOS 2.0 and later.

Declared in `CGImage.h`.

`kCGImageAlphaNoneSkipFirst`
There is no alpha channel. If the total size of the pixel is greater than the space required for the number of color components in the color space, the most significant bits are ignored.

Available in iOS 2.0 and later.

Declared in `CGImage.h`.

`kCGImageAlphaOnly`
There is no color data, only an alpha channel.

Available in iOS 2.0 and later.

Declared in `CGImage.h`.

`kCGImageAlphaNoneSkipLast`
There is no alpha channel. If the total size of the pixel is greater than the space required for the number of color components in the color space, the least significant bits are ignored. This value is equivalent to `kCGImageAlphaNone`.

Available in iOS 2.0 and later.

Declared in `CGImage.h`.

`kCGImageAlphaPremultipliedFirst`
The alpha component is stored in the most significant bits of each pixel and the color components have already been multiplied by this alpha value. For example, premultiplied ARGB.

Available in iOS 2.0 and later.

Declared in `CGImage.h`.

`kCGImageAlphaPremultipliedLast`
The alpha component is stored in the least significant bits of each pixel and the color components have already been multiplied by this alpha value. For example, premultiplied RGBA.

Available in iOS 2.0 and later.

Declared in `CGImage.h`.

Discussion
A `CGImageAlphaInfo` constant specifies (1) whether a bitmap contains an alpha channel, (2) where the alpha bits are located in the image data, and (3) whether the alpha value is premultiplied. You can obtain a `CGImageAlphaInfo` constant for an image by calling the function `CGImageGetAlphaInfo` (page 13). (You provide a `CGBitmapInfo` constant to the function `CGImageCreate` (page 7), part of which is a `CGImageAlphaInfo` constant.)

Quartz accomplishes alpha blending by combining the color components of the source image with the color components of the destination image using the linear interpolation formula, where “source” is one color component of one pixel of the new paint and “destination” is one color component of the background image.

Quartz supports premultiplied alpha only for images. You should not premultiply any other color values specified in Quartz.

Declared In
`CGImage.h`

Image Bitmap Information

Component information for a bitmap image.

```
enum {
    kCGBitmapAlphaInfoMask = 0x1F,
    kCGBitmapFloatComponents = (1 << 8),

    kCGBitmapByteOrderMask = 0x7000,
    kCGBitmapByteOrderDefault = (0 << 12),
    kCGBitmapByteOrder16Little = (1 << 12),
    kCGBitmapByteOrder32Little = (2 << 12),
    kCGBitmapByteOrder16Big = (3 << 12),
    kCGBitmapByteOrder32Big = (4 << 12)
};
typedef uint32_t CGBitmapInfo;
```

Constants

`kCGBitmapAlphaInfoMask`
The alpha information mask. Use this to extract alpha information that specifies whether a bitmap contains an alpha channel and how the alpha channel is generated.

Available in iOS 2.0 and later.

Declared in `CGImage.h`.

`kCGBitmapFloatComponents`
The components of a bitmap are floating-point values.

Available in iOS 2.0 and later.

Declared in `CGImage.h`.

`kCGBitmapByteOrderMask`
The byte ordering of pixel formats.

Available in iOS 2.0 and later.

Declared in `CGImage.h`.

`kCGBitmapByteOrderDefault`
The default byte order.

Available in iOS 2.0 and later.

Declared in `CGImage.h`.

`kCGBitmapByteOrder16Little`
16-bit, little endian format.

Available in iOS 2.0 and later.

Declared in `CGImage.h`.

`kCGBitmapByteOrder32Little`
32-bit, little endian format.

Available in iOS 2.0 and later.

Declared in `CGImage.h`.

`kCGBitmapByteOrder16Big`
16-bit, big endian format.

Available in iOS 2.0 and later.

Declared in `CGImage.h`.

`kCGBitmapByteOrder32Big`
32-bit, big endian format.

Available in iOS 2.0 and later.

Declared in `CGImage.h`.

Discussion
Applications that store pixel data in memory using ARGB format must take care in how they read data. If the code is not written correctly, it's possible to misread the data which leads to colors or alpha that appear wrong. The Quartz byte order constants specify the byte ordering of pixel formats. To specify byte ordering to Quartz use a bitwise OR operator to combine the appropriate constant with the `bitmapInfo` parameter.

Host Endian Bitmap Formats

Bit-depth constants for image bitmaps in host-endian byte order.

```
#ifdef __BIG_ENDIAN__
#define kCGBitmapByteOrder16Host kCGBitmapByteOrder16Big
#define kCGBitmapByteOrder32Host kCGBitmapByteOrder32Big
#else
#define kCGBitmapByteOrder16Host kCGBitmapByteOrder16Little
#define kCGBitmapByteOrder32Host kCGBitmapByteOrder32Little
#endif
```

Constants
`kCGBitmapByteOrder16Host`
16-bit, host endian format.

Available in iOS 2.0 and later.

Declared in `CGImage.h`.

`kCGBitmapByteOrder32Host`
32-bit, host endian format.

Available in iOS 2.0 and later.

Declared in `CGImage.h`.

Document Revision History

This table describes the changes to *CGImage Reference*.

Date	Notes
2010-08-03	Clarified that the decode matrix must also include entries for the alpha channel, if one is present.
2008-04-08	Added a cross reference to Quartz 2D Programming Guide.
2007-10-31	Updated for Mac OS X v10.5.
	All instances of the `float` data type were changed to the `CGFloat` data type.
	Added information to `CGImageCreateWithMaskingColors` (page 11).
2006-01-10	Changed CGImageBitmapInfo to CGBitmapInfo and updated the associated constants.
2005-07-07	Added documentation for Quartz constants that specify byte ordering of pixel formats.
2005-04-29	Made minor editorial corrections.
	Updated for Mac OS X v10.4.
	Added the functions `CGImageCreateWithImageInRect` (page 9), `CGImageCreateWithMask` (page 11), `CGImageCreateWithMaskingColors` (page 11), `CGImageGetBitmapInfo` (page 13), and `CGImageCreateCopy` (page 8).
	Added "`Image Bitmap Information`" (page 23) constants.
2004-08-31	Added introductory material.
2004-02-26	First version of this document. An earlier version of this information appeared in *Quartz 2D Reference*.

Document Revision History