
NSBitmapImageRep Class Reference

Graphics & Animation: 2D Drawing



2009-06-28



Apple Inc.
© 2009 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

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

Apple, the Apple logo, Cocoa, ColorSync, Mac, Mac OS, Quartz, and Spaces are trademarks of Apple Inc., registered in the United States and other countries.

NeXT is a trademark of NeXT Software, Inc., registered in the United States and other countries.

OpenGL is a registered trademark of Silicon Graphics, Inc.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY,

MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

NSBitmapImageRep Class Reference 5

Overview 5

Alpha Premultiplication 5

Tasks 6

Creating an NSBitmapImageRep Object 6

Getting Information About the Image 6

Getting Image Data 7

Producing Representations of the Image 7

Managing Compression Types 7

Loading Image Incrementally 8

Managing Pixel Values 8

Getting a Core Graphics Image 8

Managing ColorSpaces 8

Class Methods 9

getTIFFCompressionTypes:count: 9

imageRepsWithData: 9

imageRepWithData: 10

localizedNameForTIFFCompressionType: 10

representationOfImageRepsInArray:usingType:properties: 11

TIFFRepresentationOfImageRepsInArray: 11

TIFFRepresentationOfImageRepsInArray:usingCompression:factor: 12

Instance Methods 12

bitmapData 12

bitmapFormat 13

bitmapImageRepByConvertingToColorSpace:renderingIntent: 13

bitmapImageRepByRetaggingWithColorSpace: 14

bitsPerPixel 14

bytesPerPlane 15

bytesPerRow 15

canBeCompressedUsing: 16

CGImage 16

colorAtX:y: 17

colorizeByMappingGray:toColor:blackMapping:whiteMapping: 17

colorSpace 18

getBitmapDataPlanes: 18

getCompression:factor: 19

getPixel:atX:y: 20

incrementalLoadFromData:complete: 20

initWithIncrementalLoad 21

initWithBitmapDataPlanes:pixelsWide:pixelsHigh:bitsPerSample:samplesPerPixel:
hasAlpha:isPlanar:colorSpaceName:bitmapFormat:bytesPerRow:bitsPerPixel: 22

- initWithBitmapDataPlanes:pixelsWide:pixelsHigh:bitsPerSample:samplesPerPixel:
hasAlpha:isPlanar:colorSpaceName:bytesPerRow:bitsPerPixel: 24
- initWithCGImage: 26
- initWithCIImage: 27
- initWithData: 28
- initWithFocusedViewRect: 28
- isPlanar 29
- numberOfPlanes 29
- representationUsingType:properties: 29
- samplesPerPixel 30
- setColor:atX:y: 30
- setCompression:factor: 31
- setPixel:atX:y: 32
- setProperty:withValue: 32
- TIFFRepresentation 32
- TIFFRepresentationUsingCompression:factor: 33
- valueForProperty: 34
- Constants 34
 - NSImageRepLoadStatus 34
 - Bitmap image properties 36
 - NSBitmapImageFileType 38
 - NSTIFFCompression 39
 - NSBitmapFormat 40

Document Revision History 43

Index 45

NSBitmapImageRep Class Reference

Inherits from	NSImageRep : NSObject
Conforms to	NSCoding (NSImageRep) NSCopying (NSImageRep) NSObject (NSObject)
Framework	/System/Library/Frameworks/AppKit.framework
Availability	Available in Mac OS X v10.0 and later.
Companion guide	Cocoa Drawing Guide
Declared in	NSBitmapImageRep.h
Related sample code	GLSLShowpiece GLUT Image Difference Quartz EB Reducer

Overview

An `NSBitmapImageRep` is an object that can render an image from bitmap data. Bitmap data formats supported include GIF, JPEG, TIFF, PNG, and various permutations of raw bitmap data.

Alpha Premultiplication

If a coverage (alpha) plane exists, a bitmap's color components are premultiplied with it. If you modify the contents of the bitmap, you are therefore responsible for premultiplying the data. For this reason, though, if you want to manipulate the actual data, an `NSBitmapImageRep` object is not recommended for storage. If you need to work with unpremultiplied data, you should use Quartz, specifically `CGImageCreate` with `kCGImageAlphaLast`.

Note that premultiplying does not affect the output quality. Given source bitmap pixel s , destination pixel d , and alpha value a , a blend is basically

$$d' = a * s + (1 - a) * d$$

All premultiplication does is precalculate $a * s$.

Tasks

Creating an NSBitmapImageRep Object

- + [imageRepWithData:](#) (page 10)
Creates and returns an `NSBitmapImageRep` object initialized with the first image in the supplied data.
- + [imageRepsWithData:](#) (page 9)
Creates and returns an array of initialized `NSBitmapImageRep` objects corresponding to the images in the supplied data.
- [colorizeByMappingGray:toColor:blackMapping:whiteMapping:](#) (page 17)
Colorizes a grayscale image.
- [initWithBitmapDataPlanes:pixelsWide:pixelsHigh:bitsPerSample:samplesPerPixel:hasAlpha:isPlanar:colorSpaceName:bitmapFormat:bytesPerRow:bitsPerPixel:](#) (page 22)
Initializes the receiver, a newly allocated `NSBitmapImageRep` object, so it can render the specified image.
- [initWithBitmapDataPlanes:pixelsWide:pixelsHigh:bitsPerSample:samplesPerPixel:hasAlpha:isPlanar:colorSpaceName:bytesPerRow:bitsPerPixel:](#) (page 24)
Initializes the receiver, a newly allocated `NSBitmapImageRep` object, so it can render the specified image.
- [initWithCGImage:](#) (page 26)
Returns an `NSBitmapImageRep` object created from a Core Graphics image object.
- [initWithCIImage:](#) (page 27)
Returns an `NSBitmapImageRep` object created from a Core Image object.
- [initWithData:](#) (page 28)
Initializes a newly allocated `NSBitmapImageRep` from the provided data.
- [initWithFocusedViewRect:](#) (page 28)
Initializes the receiver, a newly allocated `NSBitmapImageRep` object, with bitmap data read from a rendered image.
- [initWithIncrementalLoad:](#) (page 21)
Initializes and returns the receiver, a newly allocated `NSBitmapImageRep` object, for incremental loading.

Getting Information About the Image

- [bitmapFormat](#) (page 13)
Returns the bitmap format of the receiver.
- [bitsPerPixel](#) (page 14)
Returns the number of bits allocated for each pixel in each plane of data.
- [bytesPerPlane](#) (page 15)
Returns the number of bytes in each plane or channel of data.
- [bytesPerRow](#) (page 15)
Returns the minimum number of bytes required to specify a scan line (a single row of pixels spanning the width of the image) in each data plane.

- [isPlanar](#) (page 29)
Returns YES if image data is a planar configuration and NO if its in a meshed configuration.
- [numberOfPlanes](#) (page 29)
Returns the number of separate planes image data is organized into.
- [samplesPerPixel](#) (page 30)
Returns the number of components in the data.

Getting Image Data

- [bitmapData](#) (page 12)
Returns a pointer to the bitmap data.
- [getBitmapDataPlanes:](#) (page 18)
Returns by indirection bitmap data of the receiver separated into planes.

Producing Representations of the Image

- + [TIFFRepresentationOfImageRepsInArray:](#) (page 11)
Returns a TIFF representation of the given images
- + [TIFFRepresentationOfImageRepsInArray:usingCompression:factor:](#) (page 12)
Returns a TIFF representation of the given images using a specified compression scheme and factor.
- [TIFFRepresentation](#) (page 32)
Returns a TIFF representation of the receiver.
- [TIFFRepresentationUsingCompression:factor:](#) (page 33)
Returns a TIFF representation of the image using the specified compression.
- + [representationOfImageRepsInArray:usingType:properties:](#) (page 11)
Formats the specified bitmap images using the specified storage type and properties and returns them in a data object.
- [representationUsingType:properties:](#) (page 29)
Formats the receiver's image data using the specified storage type and properties and returns it in a data object.

Managing Compression Types

- + [getTIFFCompressionTypes:count:](#) (page 9)
Returns by indirection an array of all available compression types that can be used when writing a TIFF image.
- + [localizedNameForTIFFCompressionType:](#) (page 10)
Returns an autoreleased string containing the localized name for the specified compression type.
- [canBeCompressedUsing:](#) (page 16)
Tests whether the receiver can be compressed by the specified compression scheme.
- [setCompression:factor:](#) (page 31)
Sets the receiver's compression type and compression factor.

- [getCompression:factor:](#) (page 19)
Returns by indirection the receiver's compression type and compression factor.
- [setProperty:withValue:](#) (page 32)
Sets the image's *property* to *value*.
- [valueForProperty:](#) (page 34)
Returns the value for the specified property.

Loading Image Incrementally

- [incrementalLoadFromData:complete:](#) (page 20)
Loads the current image data into an incrementally-loaded image representation and returns the current status of the image.

Managing Pixel Values

- [setColor:atX:y:](#) (page 30)
Changes the color of the pixel at the specified coordinates.
- [colorAtX:y:](#) (page 17)
Returns the color of the pixel at the specified coordinates.
- [setPixel:atX:y:](#) (page 32)
Sets the receiver's pixel at the specified coordinates to the specified raw pixel values.
- [getPixel:atX:y:](#) (page 20)
Returns by indirection the pixel data for the specified location in the receiver.

Getting a Core Graphics Image

- [CGImage](#) (page 16)
Returns a Core Graphics image object from the receiver's current bitmap data.

Managing ColorSpaces

- [bitmapImageRepByConvertingToColorSpace:renderingIntent:](#) (page 13)
Converts the image rep to the specified colorspace
- [bitmapImageRepByRetaggingWithColorSpace:](#) (page 14)
Changes the colorSpace tag of the receiver.
- [colorSpace](#) (page 18)
Returns the image rep's colorSpace

Class Methods

getTIFFCompressionTypes:count:

Returns by indirection an array of all available compression types that can be used when writing a TIFF image.

```
+ (void)getTIFFCompressionTypes:(const NSTIFFCompression **)list count:(NSInteger *)numTypes
```

Parameters

list

On return, a C array of `NSTIFFCompression` constants. This array belongs to the `NSBitmapImageRep` class; it shouldn't be freed or altered. See “[Constants](#)” (page 34) for the supported TIFF compression types.

numTypes

The number of constants in list.

Discussion

Note that not all compression types can be used for all images: `NSTIFFCompressionNEXT` can be used only to retrieve image data. Because future releases may include other compression types, always use this method to get the available compression types—for example, when you implement a user interface for selecting compression types.

Availability

Available in Mac OS X v10.0 and later.

See Also

- + [localizedNameForTIFFCompressionType:](#) (page 10)
- [canBeCompressedUsing:](#) (page 16)

Declared In

`NSBitmapImageRep.h`

imageRepsWithData:

Creates and returns an array of initialized `NSBitmapImageRep` objects corresponding to the images in the supplied data.

```
+ (NSArray *)imageRepsWithData:(NSData *)bitmapData
```

Parameters

bitmapData

A data object containing one or more bitmapped images or `nil` if the class is unable to create an image representation. The *bitmapData* parameter can contain data in any supported bitmap format.

Return Value

An array of `NSBitmapImageRep` instances or an empty array if the class is unable to create any image representations.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSBitmapImageRep.h

imageRepWithData:

Creates and returns an `NSBitmapImageRep` object initialized with the first image in the supplied data.

```
+ (id)imageRepWithData:(NSData *)bitmapData
```

Parameters

bitmapData

A data object containing one or more bitmapped images. The *bitmapData* parameter can contain data in any supported bitmap format.

Return Value

An `NSBitmapImageRep` instance or `nil` if the class is unable to create an image representation.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

OpenCL Procedural Grass and Terrain Example

SpecialPictureProtocol

Declared In

NSBitmapImageRep.h

localizedNameForTIFFCompressionType:

Returns an autoreleased string containing the localized name for the specified compression type.

```
+ (NSString *)localizedNameForTIFFCompressionType:(NSTIFFCompression)compression
```

Parameters

compression

A TIFF compression type. `NSTIFFCompression` types are described in “Constants” (page 34).

Return Value

A localized name for *compression* or `nil` if *compression* is unrecognized.

Discussion

When implementing a user interface for selecting TIFF compression types, use [getTIFFCompressionTypes:count:](#) (page 9) to get the list of supported compression types, then use this method to get the localized names for each compression type.

Availability

Available in Mac OS X v10.0 and later.

See Also

+ [getTIFFCompressionTypes:count:](#) (page 9)

Declared In

NSBitmapImageRep.h

representationOfImageRepsInArray:usingType:properties:

Formats the specified bitmap images using the specified storage type and properties and returns them in a data object.

```
+ (NSData *)representationOfImageRepsInArray:(NSArray *)imageReps
    usingType:(NSBitmapImageFileType)storageType properties:(NSDictionary
*)properties
```

Parameters

imageReps

An array of NSBitmapImageRep objects.

storageType

An enum constant specifying a file type for bitmap images. It can be NSBMPFileType, NSGIFFFileType, NSJPEGFileType, NSPNGFileType, or NSTIFFFileType.

properties

A dictionary that contains key-value pairs specifying image properties. These string constants used as keys and the valid values are described in “[Bitmap image properties](#)” (page 36).

Return Value

A data object containing the bitmap image data in the specified format. You can write this data to a file or use it to create a new NSBitmapImageRep object.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSBitmapImageRep.h

TIFFRepresentationOfImageRepsInArray:

Returns a TIFF representation of the given images

```
+ (NSData *)TIFFRepresentationOfImageRepsInArray:(NSArray *)array
```

Parameters

array

An array containing objects representing bitmap image representations.

Return Value

A data object containing a TIFF image representation.

Discussion

This method uses the compression returned by [getCompression:factor:](#) (page 19) (if applicable). If a problem is encountered during generation of the TIFF, this method raises an `NSTIFFException` or an `NSBadBitmapParametersException`.

Availability

Available in Mac OS X v10.0 and later.

See Also

- [TIFFRepresentation](#) (page 32)

Declared In

NSBitmapImageRep.h

TIFFRepresentationOfImageRepsInArray:usingCompression:factor:

Returns a TIFF representation of the given images using a specified compression scheme and factor.

```
+ (NSData *)TIFFRepresentationOfImageRepsInArray:(NSArray *)array
    usingCompression:(NSTIFFCompression)compression factor:(float)factor
```

Parameters*array*

An array containing objects representing bitmap image representations.

*compression*An enum constant that represents a TIFF data-compression scheme. Legal values for *compression* can be found in NSBitmapImageRep.h and are described in “Constants” (page 34).*factor*

A float value that provides a hint for those compression types that implement variable compression ratios.

Currently only JPEG compression uses a compression factor. JPEG compression in TIFF files is not supported, and *factor* is ignored.**Return Value**

A data object containing a TIFF image representation.

DiscussionIf the specified compression isn't applicable, no compression is used. If a problem is encountered during generation of the TIFF, the method raises an `NSTIFFException` or an `NSBadBitmapParametersException`.**Availability**

Available in Mac OS X v10.0 and later.

See Also- [TIFFRepresentationUsingCompression:factor:](#) (page 33)**Declared In**

NSBitmapImageRep.h

Instance Methods

bitmapData

Returns a pointer to the bitmap data.

- (unsigned char *)bitmapData

Discussion

If the data is planar, returns a pointer to the first plane.

Availability

Available in Mac OS X v10.0 and later.

See Also

- [getPixel:atX:y:](#) (page 20)
- [getBitmapDataPlanes:](#) (page 18)

Related Sample Code

Image Difference
LayerBackedOpenGLView
NSOpenGL Fullscreen
Quartz EB
SimpleImageFilter

Declared In

NSBitmapImageRep.h

bitmapFormat

Returns the bitmap format of the receiver.

- (NSBitmapFormat)bitmapFormat

Discussion

Returns 0 by default. The return value can indicate several different attributes, which are described in [“Constants”](#) (page 34).

Availability

Available in Mac OS X v10.4 and later.

See Also

- [bytesPerRow](#) (page 15)

Declared In

NSBitmapImageRep.h

bitmapImageRepByConvertingToColorSpace:renderingIntent:

Converts the image rep to the specified colorspace

- (NSBitmapImageRep *)bitmapImageRepByConvertingToColorSpace:(NSColorSpace *)targetSpace renderingIntent:(NSColorRenderingIntent)renderingIntent

Parameters

targetSpace

The new colorSpace

renderingIntent

The rendering intent specifies how to handle colors that are not located within the target color space. The supported values are NSColorRenderingIntent.

Return Value

An `NSBitmapImageRep`, or `nil`, if the conversion fails. If the original `NSBitmapImageRep` already uses that `colorSpace`, it is returned as is.

Availability

Available in Mac OS X v10.6 and later.

See Also

- [bitmapImageRepByRetaggingWithColorSpace:](#) (page 14)
- [colorSpace](#) (page 18)

Declared In

`NSBitmapImageRep.h`

bitmapImageRepByRetaggingWithColorSpace:

Changes the `colorSpace` tag of the receiver.

```
- (NSBitmapImageRep *)bitmapImageRepByRetaggingWithColorSpace:(NSColorSpace
 *)newSpace
```

Parameters

newSpace

The desired `colorSpace`.

Return Value

An `NSBitmapImageRep`, or `nil`, if the conversion fails. If the original `NSBitmapImageRep` already uses that `colorSpace`, it is returned as is.

Discussion

This method will definitely fail if you pass a `colorSpace` that has a different color space model than the receiver. That is, if your original image is sRGB, you can only retag with some other RGB colorspace.

Availability

Available in Mac OS X v10.6 and later.

See Also

- [bitmapImageRepByRetaggingWithColorSpace:](#) (page 14)
- [colorSpace](#) (page 18)

Declared In

`NSBitmapImageRep.h`

bitsPerPixel

Returns the number of bits allocated for each pixel in each plane of data.

```
- (NSInteger)bitsPerPixel
```

Discussion

This number is normally equal to the number of bits per sample or, if the data is in meshed configuration, the number of bits per sample times the number of samples per pixel. It can be explicitly set to another value (in [initWithBitmapDataPlanes:pixelWide:pixelHigh:bitsPerSample:samplesPerPixel:hasAlpha:isPlanar:colorSpaceName:bytesPerRow:bitsPerPixel:](#) (page 24)) in case extra memory is allocated for each pixel. This may be the case, for example, if pixel data is aligned on byte boundaries.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Quartz EB

Declared In

NSBitmapImageRep.h

bytesPerPlane

Returns the number of bytes in each plane or channel of data.

- (NSInteger)bytesPerPlane

Discussion

This number is calculated from the number of bytes per row and the height of the image.

Availability

Available in Mac OS X v10.0 and later.

See Also

- [bytesPerRow](#) (page 15)

Declared In

NSBitmapImageRep.h

bytesPerRow

Returns the minimum number of bytes required to specify a scan line (a single row of pixels spanning the width of the image) in each data plane.

- (NSInteger)bytesPerRow

Discussion

If not explicitly set to another value (in [initWithBitmapDataPlanes:pixelWide:pixelHigh:bitsPerSample:samplesPerPixel:hasAlpha:isPlanar:colorSpaceName:bytesPerRow:bitsPerPixel:](#) (page 24)), this number will be figured from the width of the image, the number of bits per sample, and, if the data is in a meshed configuration, the number of samples per pixel. It can be set to another value to indicate that each row of data is aligned on word or other boundaries.

Availability

Available in Mac OS X v10.0 and later.

See Also

- [bytesPerPlane](#) (page 15)

Related Sample Code

FunHouse

GLUT

Quartz EB

Vertex Optimization

VertexPerformanceDemo

Declared In

NSBitmapImageRep.h

canBeCompressedUsing:

Tests whether the receiver can be compressed by the specified compression scheme.

- (BOOL)canBeCompressedUsing:(NSTIFFCompression)*compression*

Parameters

compression

A TIFF compression type. NSTIFFCompression types are defined in “Constants” (page 34).

Return Value

YES if the receiver's data matches *compression* with this type, NO if the data doesn't match *compression* or if *compression* is unsupported..

Discussion

Legal values for *compression* can be found in NSBitmapImageRep.h and are described in TIFF Compression in NSBitmapImageReps. This method returns YES if the receiver's data matches *compression*; for example, if *compression* is NSTIFFCompressionCCITTFAX3, then the data must be 1 bit per sample and 1 sample per pixel.

Availability

Available in Mac OS X v10.0 and later.

See Also

+ [getTIFFCompressionTypes:count:](#) (page 9)

Declared In

NSBitmapImageRep.h

CGImage

Returns a Core Graphics image object from the receiver's current bitmap data.

- (CGImageRef)CGImage

Return Value

Returns an autoreleased CGImageRef opaque type based on the receiver's current bitmap data.

Discussion

The returned `CGImageRef` has pixel dimensions that are identical to the receiver's. This method might return a preexisting `CGImageRef` opaque type or create a new one. If the receiver is later modified, subsequent invocations of this method might return different `CGImageRef` opaque types.

Availability

Available in Mac OS X version 10.5.

See Also

- [initWithCGImage:](#) (page 26)

Declared In

NSBitmapImageRep.h

colorAtX:y:

Returns the color of the pixel at the specified coordinates.

```
- (NSColor *)colorAtX:(NSInteger)x y:(NSInteger)y
```

Parameters

x

The x-axis coordinate.

y

The y-axis coordinate.

Return Value

A color object representing the color at the specified coordinates.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [setColor:atX:y:](#) (page 30)

Related Sample Code

RadiantColorPicker

Declared In

NSBitmapImageRep.h

colorizeByMappingGray:toColor:blackMapping:whiteMapping:

Colorizes a grayscale image.

```
- (void)colorizeByMappingGray:(CGFloat)midPoint toColor:(NSColor *)midPointColor
    blackMapping:(NSColor *)shadowColor whiteMapping:(NSColor *)lightColor
```

Parameters

midPoint

A float value representing the midpoint of the grayscale image.

midPointColor

A color object representing the midpoint of the color to map the image to.

shadowColor

A color object representing the black mapping to use for shadows.

lightColor

A color object representing the white mapping to be used in the image.

Discussion

This method maps the receiver such that:

```
Gray value of midPoint -> midPointColor;
black -> shadowColor;
white -> lightColor.
```

It works on images with 8-bit SPP, and thus supports either 8-bit gray or 24-bit color (with optional alpha).

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSBitmapImageRep.h

colorSpace

Returns the image rep's colorSpace

```
- (NSColorSpace *)colorSpace
```

Return Value

The colorSpace.

Availability

Available in Mac OS X v10.6 and later.

See Also

- [bitmapImageRepByRetaggingWithColorSpace:](#) (page 14)
- [bitmapImageRepByConvertingToColorSpace:renderingIntent:](#) (page 13)

Declared In

NSBitmapImageRep.h

getBitmapDataPlanes:

Returns by indirection bitmap data of the receiver separated into planes.

```
- (void)getBitmapDataPlanes:(unsigned char **)data
```

Parameters*data*

On return, a C array of five character pointers. If the bitmap data is in planar configuration, each pointer will be initialized to point to one of the data planes. If there are less than five planes, the remaining pointers will be set to NULL. If the bitmap data is in meshed configuration, only the first pointer will be initialized; the others will be NULL.

Discussion

Color components in planar configuration are arranged in the expected order—for example, red before green before blue for RGB color. All color planes precede the coverage plane. If a coverage plane exists, the bitmap's color components are premultiplied with it. If you modify the contents of the bitmap, you are responsible for premultiplying the data.

Availability

Available in Mac OS X v10.0 and later.

See Also

- [isPlanar](#) (page 29)

- [initWithBitmapDataPlanes:pixelSwide:pixelHigh:bitsPerSample:samplesPerPixel:hasAlpha:isPlanar:colorSpaceName:bitmapFormat:bytesPerRow:bitsPerPixel:](#) (page 22)

- [initWithBitmapDataPlanes:pixelSwide:pixelHigh:bitsPerSample:samplesPerPixel:hasAlpha:isPlanar:colorSpaceName:bytesPerRow:bitsPerPixel:](#) (page 24)

Declared In

NSBitmapImageRep.h

getCompressionFactor:

Returns by indirection the receiver's compression type and compression factor.

```
- (void)getCompression:(NSTIFFCompression *)compression factor:(float *)factor
```

Parameters*compression*

On return, an enum constant that represents the compression type used on the data; it corresponds to one of the values returned by the class method [getTIFFCompressionTypes:count:](#) (page 9).

factor

A float value that is specific to the compression type. Many types of compression don't support varying degrees of compression and thus ignore *factor*. JPEG compression allows a compression factor ranging from 0.0 to 1.0, with 0.0 being the lowest and 1.0 being the highest.

Discussion

Use this method to get information on the compression type for the source image data.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSBitmapImageRep.h

getPixel:atX:y:

Returns by indirection the pixel data for the specified location in the receiver.

```
- (void)getPixel:(NSUInteger[])pixelData atX:(NSInteger)x y:(NSInteger)y
```

Parameters

pixelData

On return, an array of integers containing raw pixel data in the appropriate order for the receiver's [bitmapFormat](#) (page 13). Smaller integer samples, such as 4-bit, are returned as an integer. Floating point values are cast to integer values and returned.

x

The x-axis coordinate of the pixel.

y

The y-axis coordinate of the pixel.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [setPixel:atX:y:](#) (page 32)

Declared In

NSBitmapImageRep.h

incrementalLoadFromData:complete:

Loads the current image data into an incrementally-loaded image representation and returns the current status of the image.

```
- (NSInteger)incrementalLoadFromData:(NSData *)data complete:(BOOL)complete
```

Parameters

data

A data object that contains the image to be loaded.

complete

YES if the image is entirely downloaded, NO otherwise.

Return Value

An integer constant indicating the status of the image during the load operation. See the discussion for details.

Discussion

After initializing the receiver with [initWithIncrementalLoad](#) (page 21), you should call this method to incrementally load the image. Call this method each time new data becomes available. Always pass the entire image data buffer in *data*, not just the newest data, because the image decompressor may need the original data in order to backtrack. This method will block until the data is decompressed; it will decompress as much of the image as possible based on the length of the data. The image rep does not retain *data*, so you must ensure that *data* is not released for the duration of this method call. Pass NO for *complete* until the entire image is downloaded, at which time you should pass YES. You should also pass YES for *complete* if you have only partially downloaded the data, but cannot finish the download.

This method returns `NSImageRepLoadStatusUnknownType` if you did not pass enough data to determine the image format; you should continue to invoke this method with additional data.

This method returns `NSImageRepLoadStatusReadingHeader` if it has enough data to determine the image format, but needs more data to determine the size and depth and other characteristics of the image. You should continue to invoke this method with additional data.

This method returns `NSImageRepLoadStatusWillNeedAllData` if the image format does not support incremental loading or the Application Kit does not yet implement incremental loading for the image format. You may continue to invoke this method in this case, but until you pass YES for *complete*, this method will continue to return `NSImageRepLoadStatusWillNeedAllData`, and will perform no decompression. Once you pass YES, the image will be decompressed and one of the final three status messages will be returned.

If the image format does support incremental loading, then once enough data has been read, the image is decompressed from the top down a row at a time. In this case, instead of a status value, this method returns the number of pixel rows that have been decompressed, starting from the top of the image. You can use this information to draw the part of the image that is valid. The rest of the image is filled with opaque white. Note that if the image is progressive (as in a progressive JPEG or 2D interlaced PNG), this method may quickly return the full height of the image, but the image may still be loading, so do not use this return value as an indication of how much of the image remains to be decompressed.

If an error occurred while decompressing, this method returns `NSImageRepLoadStatusInvalidData`. If *complete* is YES but not enough data was available for decompression, `NSImageRepLoadStatusUnexpectedEOF` is returned. If enough data has been provided (regardless of the *complete* flag), then `NSImageRepLoadStatusCompleted` is returned. When any of these three status results are returned, this method has adjusted the `NSBitmapImageRep` so that `pixelsHigh` and `size`, as well as the bitmap data, only contains the pixels that are valid, if any.

To cancel decompression, just pass in the existing data or `nil` and YES for *complete*. This method stops decompression immediately, adjusts the image size, and returns `NSImageRepLoadStatusUnexpectedEOF`. This method returns `NSImageRepLoadStatusCompleted` if you call it after receiving any error results (`NSImageRepLoadStatusInvalidData` or `NSImageRepLoadStatusUnexpectedEOF`) or if you call it on an `NSBitmapImageRep` that was not initialized with [initForIncrementalLoad](#) (page 21).

Availability

Available in Mac OS X v10.2 and later.

See Also

- [initForIncrementalLoad](#) (page 21)

Declared In

`NSBitmapImageRep.h`

initForIncrementalLoad

Initializes and returns the receiver, a newly allocated `NSBitmapImageRep` object, for incremental loading.

- (id)initForIncrementalLoad

Discussion

The receiver returns itself after setting its size and data buffer to zero. You can then call [incrementalLoadFromData:complete:](#) (page 20) to incrementally add image data.

Availability

Available in Mac OS X v10.2 and later.

See Also

- [incrementalLoadFromData:complete:](#) (page 20)

Declared In

NSBitmapImageRep.h

initWithBitmapDataPlanes:pixelsWide:pixelsHigh:bitsPerSample:samplesPerPixel:hasAlpha:isPlanar:colorSpaceName:bitmapFormat:bytesPerRow:bitsPerPixel:

Initializes the receiver, a newly allocated NSBitmapImageRep object, so it can render the specified image.

```
- (id)initWithBitmapDataPlanes:(unsigned char **)planes pixelsWide:(NSInteger)width
    pixelsHigh:(NSInteger)height bitsPerSample:(NSInteger)bps
    samplesPerPixel:(NSInteger)spp hasAlpha:(BOOL)alpha isPlanar:(BOOL)isPlanar
    colorSpaceName:(NSString *)colorSpaceName
    bitmapFormat:(NSBitmapFormat)bitmapFormat bytesPerRow:(NSInteger)rowBytes
    bitsPerPixel:(NSInteger)pixelBits
```

Parameters

planes

An array of character pointers, each of which points to a buffer containing raw image data. If the data is in planar configuration, each buffer holds one component—one plane—of the data. Color planes are arranged in the standard order—for example, red before green before blue for RGB color. All color planes precede the coverage plane. If a coverage plane exists, the bitmap's color components must be premultiplied with it. If the data is in meshed configuration (that is, *isPlanar* is NO), only the first buffer is read.

If *planes* is NULL or an array of NULL pointers, this method allocates enough memory to hold the image described by the other arguments. You can then obtain pointers to this memory (with the [getPixel:atX:y:](#) (page 20) or [bitmapData](#) (page 12) method) and fill in the image data. In this case, the allocated memory will belong to the object and will be freed when it's freed.

If *planes* is not NULL and the array contains at least one data pointer, the returned object will only reference the image data; it will not copy it. The object treats the image data in the buffers as immutable and will not attempt to alter it. When the object itself is freed, it will not attempt to free the buffers.

width

The width of the image in pixels. This value must be greater than 0.

height

The height of the image in pixels. This value must be greater than 0.

bps

The number of bits used to specify one pixel in a single component of the data. All components are assumed to have the same bits per sample. *bps* should be one of these values: 1, 2, 4, 8, 12, or 16.

spp

The number of data components, or samples per pixel. This value includes both color components and the coverage component (alpha), if present. Meaningful values range from 1 through 5. An image with cyan, magenta, yellow, and black (CMYK) color components plus a coverage component would have an *spp* of 5; a grayscale image that lacks a coverage component would have an *spp* of 1.

alpha

YES if one of the components counted in the number of samples per pixel (*spp*) is a coverage (alpha) component, and NO if there is no coverage component. If YES, the color components in the bitmap data must be premultiplied with their coverage component.

isPlanar

YES if the data components are laid out in a series of separate “planes” or channels (“planar configuration”) and NO if component values are interwoven in a single channel (“meshed configuration”). If NO, only the first buffer of planes is read.

For example, in meshed configuration, the red, green, blue, and coverage values for the first pixel of an image would precede the red, green, blue, and coverage values for the second pixel, and so on. In planar configuration, red values for all the pixels in the image would precede all green values, which would precede all blue values, which would precede all coverage values.

colorSpaceName

A string constant that indicates how data values are to be interpreted. It should be one of the following values:

- NSCalibratedWhiteColorSpace
- NSCalibratedBlackColorSpace
- NSCalibratedRGBColorSpace
- NSDeviceWhiteColorSpace
- NSDeviceBlackColorSpace
- NSDeviceRGBColorSpace
- NSDeviceCMYKColorSpace
- NSNamedColorSpace
- NSCustomColorSpace

If *bps* is 12, you cannot specify the monochrome color space.

bitmapFormat

An integer that specifies the ordering of the bitmap components. It is a mask created by combining the NSBitmapFormat constants NSAlphaFirstBitmapFormat, NSAlphaNonpremultipliedBitmapFormat and NSFloatingPointSamplesBitmapFormat using the C bitwise OR operator.

rowBytes

The number of bytes that are allocated for each scan line in each plane of data. A scan line is a single row of pixels spanning the width of the image.

Normally, *rowBytes* can be figured from the width of the image, the number of bits per pixel in each sample (*bps*), and, if the data is in a meshed configuration, the number of samples per pixel (*spp*). However, if the data for each row is aligned on word or other boundaries, it may have been necessary to allocate more memory for each row than there is data to fill it. *rowBytes* lets the object know whether that’s the case.

If you pass in a *rowBytes* value of 0, the bitmap data allocated may be padded to fall on long word or larger boundaries for performance. If your code wants to advance row by row, use [bytesPerRow](#) (page 15) and do not assume the data is packed. Passing in a non-zero value allows you to specify exact row advances.

pixelBits

This integer value informs `NSBitmapImageRep` how many bits are actually allocated per pixel in each plane of data. If the data is in planar configuration, this normally equals *bps* (bits per sample). If the data is in meshed configuration, it normally equals *bps* times *spp* (samples per pixel). However, it's possible for a pixel specification to be followed by some meaningless bits (empty space), as may happen, for example, if pixel data is aligned on byte boundaries. `NSBitmapImageRep` supports only a limited number of *pixelBits* values (other than the default): for RGB images with 4 *bps*, *pixelBits* may be 16; for RGB images with 8 *bps*, *pixelBits* may be 32. The legal values for *pixelBits* are system dependent.

If you specify 0 for this parameter, the object interprets the number of bits per pixel using the values in the *bps* and *spp* parameters, as described in the preceding paragraph, without any meaningless bits.

Return Value

An initialized `NSBitmapImageRep` object or `nil` if the object cannot be initialized.

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

Quartz Composer Offline Rendering

Declared In

`NSBitmapImageRep.h`

`initWithBitmapDataPlanes:pixelsWide:pixelsHigh:bitsPerSample:samplesPerPixel:hasAlpha:isPlanar:colorSpaceName:bytesPerRow:bitsPerPixel:`

Initializes the receiver, a newly allocated `NSBitmapImageRep` object, so it can render the specified image.

```
- (id)initWithBitmapDataPlanes:(unsigned char **)planes pixelsWide:(NSInteger)width
    pixelsHigh:(NSInteger)height bitsPerSample:(NSInteger)bps
    samplesPerPixel:(NSInteger)spp hasAlpha:(BOOL)alpha isPlanar:(BOOL)isPlanar
    colorSpaceName:(NSString *)colorSpaceName bytesPerRow:(NSInteger)rowBytes
    bitsPerPixel:(NSInteger)pixelBits
```

Parameters*planes*

An array of character pointers, each of which points to a buffer containing raw image data. If the data is in planar configuration, each buffer holds one component—one plane—of the data. Color planes are arranged in the standard order—for example, red before green before blue for RGB color. All color planes precede the coverage plane. If a coverage plane exists, the bitmap's color components must be premultiplied with it. If the data is in meshed configuration (that is, *isPlanar* is NO), only the first buffer is read.

If *planes* is NULL or an array of NULL pointers, this method allocates enough memory to hold the image described by the other arguments. You can then obtain pointers to this memory (with the [getPixel:atX:y:](#) (page 20) or [bitmapData](#) (page 12) method) and fill in the image data. In this case, the allocated memory will belong to the object and will be freed when it's freed.

If *planes* is not NULL and the array contains at least one data pointer, the returned object will only reference the image data; it will not copy it. The object treats the image data in the buffers as immutable and will not attempt to alter it. When the object itself is freed, it will not attempt to free the buffers.

width

The width of the image in pixels. This value must be greater than 0.

height

The height of the image in pixels. This value must be greater than 0.

bps

The number of bits used to specify one pixel in a single component of the data. All components are assumed to have the same bits per sample. *bps* should be one of these values: 1, 2, 4, 8, 12, or 16.

spp

The number of data components, or samples per pixel. This value includes both color components and the coverage component (alpha), if present. Meaningful values range from 1 through 5. An image with cyan, magenta, yellow, and black (CMYK) color components plus a coverage component would have an *spp* of 5; a grayscale image that lacks a coverage component would have an *spp* of 1.

alpha

YES if one of the components counted in the number of samples per pixel (*spp*) is a coverage (alpha) component, and NO if there is no coverage component. If YES, the color components in the bitmap data must be premultiplied with their coverage component.

isPlanar

YES if the data components are laid out in a series of separate “planes” or channels (“planar configuration”) and NO if component values are interwoven in a single channel (“meshed configuration”). If NO, only the first buffer of planes is read.

For example, in meshed configuration, the red, green, blue, and coverage values for the first pixel of an image would precede the red, green, blue, and coverage values for the second pixel, and so on. In planar configuration, red values for all the pixels in the image would precede all green values, which would precede all blue values, which would precede all coverage values.

colorSpaceName

A string constant that indicates how data values are to be interpreted. It should be one of the following values:

- NSCalibratedWhiteColorSpace
- NSCalibratedBlackColorSpace
- NSCalibratedRGBColorSpace
- NSDeviceWhiteColorSpace
- NSDeviceBlackColorSpace
- NSDeviceRGBColorSpace
- NSDeviceCMYKColorSpace
- NSNamedColorSpace
- NSCustomColorSpace

If *bps* is 12, you cannot specify the monochrome color space.

rowBytes

The number of bytes that are allocated for each scan line in each plane of data. A scan line is a single row of pixels spanning the width of the image.

Normally, *rowBytes* can be figured from the width of the image, the number of bits per pixel in each sample (*bps*), and, if the data is in a meshed configuration, the number of samples per pixel (*spp*). However, if the data for each row is aligned on word or other boundaries, it may have been necessary to allocate more memory for each row than there is data to fill it. *rowBytes* lets the object know whether that's the case.

If you pass in a *rowBytes* value of 0, the bitmap data allocated may be padded to fall on long word or larger boundaries for performance. If your code wants to advance row by row, use [bytesPerRow](#) (page 15) and do not assume the data is packed. Passing in a non-zero value allows you to specify exact row advances.

pixelBits

This integer value informs NSBitmapImageRep how many bits are actually allocated per pixel in each plane of data. If the data is in planar configuration, this normally equals *bps* (bits per sample). If the data is in meshed configuration, it normally equals *bps* times *spp* (samples per pixel). However, it's possible for a pixel specification to be followed by some meaningless bits (empty space), as may happen, for example, if pixel data is aligned on byte boundaries. NSBitmapImageRep supports only a limited number of *pixelBits* values (other than the default): for RGB images with 4 *bps*, *pixelBits* may be 16; for RGB images with 8 *bps*, *pixelBits* may be 32. The legal values for *pixelBits* are system dependent.

If you specify 0 for this parameter, the object interprets the number of bits per pixel using the values in the *bps* and *spp* parameters, as described in the preceding paragraph, without any meaningless bits.

Return Value

An initialized NSBitmapImageRep object or nil if the object cannot be initialized.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AnimatedTableView

CocoaVideoFrameToNSImage

ColorMatching

Monochrome Image

RadiantColorPicker

Declared In

NSBitmapImageRep.h

initWithCGImage:

Returns an NSBitmapImageRep object created from a Core Graphics image object.

```
- (id)initWithCGImage:(CGImageRef)cgImage
```

Parameters

cgImage

A Core Graphics image object (an opaque type) from which to create the receiver. This opaque type is retained.

Return Value

An `NSBitmapImageRep` object initialized from the contents of the Core Graphics image or `nil` if the `NSBitmapImageRep` couldn't be created.

Discussion

If you use this method, you should treat the resulting bitmap `NSBitmapImageRep` object as read only. Because it only retains the value in the `cgImage` parameter, rather than unpacking the data, accessing the pixel data requires the creation of a copy of that data in memory. Changes to that data are not saved back to the Core Graphics image.

Availability

Available in Mac OS X v10.5 and later.

See Also

- [CGImage](#) (page 16)

Declared In

`NSBitmapImageRep.h`

initWithCIImage:

Returns an `NSBitmapImageRep` object created from a Core Image object.

```
- (id)initWithCIImage:(CIImage *)ciImage
```

Parameters

ciImage

A Core Image object whose contents are to be copied to the receiver. This image rectangle must be of a finite size.

Return Value

An `NSBitmapImageRep` object initialized from the contents of the Core Image (`CIImage`) object or `nil` if the `NSBitmapImageRep` couldn't be created.

Discussion

The image in the *ciImage* parameter must be fully rendered before the receiver can be initialized. If you specify an object whose rendering was deferred (and thus does not have any pixels available now), this method forces the image to be rendered immediately. Rendering the image could result in a performance penalty if the image has a complex rendering chain or accelerated rendering hardware is not available. By the time this method returns, however, the resultant `NSBitmapImageRep` object can have its raw pixel data inspected, can be put on the pasteboard, and can be encoded to any of the standard image formats that `NSBitmapImageRep` supports (JPEG, TIFF, and so on.)

If you pass in a `CIImage` object whose extents are not finite, this method raises an exception.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `initWithBitmapImageRep:(CIImage)`

Declared In

`NSBitmapImageRep.h`

initWithData:

Initializes a newly allocated `NSBitmapImageRep` from the provided data.

```
- (id)initWithData:(NSData *)bitmapData
```

Parameters

bitmapData

A data object containing image data. The contents of *bitmapData* can be any supported bitmap format. For TIFF data, the `NSBitmapImageRep` is initialized from the first header and image data found in *bitmapData*.

Return Value

Returns an initialized `NSBitmapImageRep` if the initialization was successful or `nil` if it was unable to interpret the contents of *bitmapData*.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSBitmapImageRep.h`

initWithFocusedViewRect:

Initializes the receiver, a newly allocated `NSBitmapImageRep` object, with bitmap data read from a rendered image.

```
- (id)initWithFocusedViewRect:(NSRect)rect
```

Parameters

rect

A rectangle that specifies an area of the current window in the current coordinate system.

Return Value

Returns the initialized object or `nil` if for any reason the new object can't be initialized.

Discussion

This method uses imaging operators to read the image data into a buffer; the object is then created from that data. The object is initialized with information about the image obtained from the window server.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Color Sampler

FunHouse

GLChildWindowDemo

OpenGL Screensaver

Reducer

Declared In

`NSBitmapImageRep.h`

isPlanar

Returns YES if image data is a planar configuration and NO if its in a meshed configuration.

- (BOOL)isPlanar

Discussion

In a planar configuration, the image data is segregated into a separate plane for each color and coverage component. In a meshed configuration, the data is integrated into a single plane.

Availability

Available in Mac OS X v10.0 and later.

See Also

- [samplesPerPixel](#) (page 30)

Declared In

NSBitmapImageRep.h

numberOfPlanes

Returns the number of separate planes image data is organized into.

- (NSInteger)numberOfPlanes

Discussion

This number is the number of samples per pixel if the data has a separate plane for each component ([isPlanar](#) (page 29) returns YES) and 1 if the data is meshed ([isPlanar](#) (page 29) returns NO).

Availability

Available in Mac OS X v10.0 and later.

See Also

- [samplesPerPixel](#) (page 30)
- [hasAlpha](#) (NSImageRep)
- [bitsPerSample](#) (NSImageRep)

Declared In

NSBitmapImageRep.h

representationUsingType:properties:

Formats the receiver's image data using the specified storage type and properties and returns it in a data object.

- (NSData *)representationUsingType:(NSBitmapImageFileType)storageType
properties:(NSDictionary *)properties

Parameters

storageType

An enum constant specifying a file type for bitmap images. It can be NSBMPPFileType, NSGIFFFileType, NSJPEGFileType, NSPNGFileType, or NSTIFFFileType.

properties

A dictionary that contains key-value pairs specifying image properties. These string constants used as keys and the valid values are described in “[Bitmap image properties](#)” (page 36).

Return Value

A data object containing the receiver’s image data in the specified format. You can write this data to a file or use it to create a new NSBitmapImageRep object.

Availability

Available in Mac OS X v10.0 and later.

See Also

- [TIFFRepresentation](#) (page 32)
- [TIFFRepresentationUsingCompression:factor:](#) (page 33)
- [TIFFRepresentation \(NSImage\)](#)
- [TIFFRepresentationUsingCompression:factor: \(NSImage\)](#)

Related Sample Code

Reducer

SpecialPictureProtocol

Declared In

NSBitmapImageRep.h

samplesPerPixel

Returns the number of components in the data.

- (NSInteger)samplesPerPixel

Discussion

The returned value includes both color components and the coverage component, if present.

Availability

Available in Mac OS X v10.0 and later.

See Also

- [hasAlpha \(NSImageRep\)](#)
- [bitsPerSample \(NSImageRep\)](#)

Related Sample Code

Image Difference

Declared In

NSBitmapImageRep.h

setColor:atX:y:

Changes the color of the pixel at the specified coordinates.

- (void)setColor:(NSColor *)color atX:(NSInteger)x y:(NSInteger)y

Parameters*color*

A color object representing the color to be set.

x

The x-axis coordinate of the pixel.

y

The y-axis coordinate of the pixel.

Availability

Available in Mac OS X v10.4 and later.

See Also- [colorAtX:y:](#) (page 17)**Related Sample Code**

RadiantColorPicker

Declared In

NSBitmapImageRep.h

setCompression:factor:

Sets the receiver's compression type and compression factor.

- (void)setCompression:(NSTIFFCompression)*compression* factor:(float)*factor***Parameters***compression*An enum constant that identifies one of the supported compression types as described in ["Constants"](#) (page 34).*factor*A floating point value that is specific to the compression type. Many types of compression don't support varying degrees of compression and thus ignore *factor*. JPEG compression allows a compression factor ranging from 0.0 to 1.0, with 0.0 being the lowest and 1.0 being the highest.**Discussion**

When an NSBitmapImageRep is created, the instance stores the compression type and factor for the source data. [TIFFRepresentation](#) (page 32) and [TIFFRepresentationOfImageRepsInArray:](#) (page 11) (class method) try to use the stored compression type and factor. Use this method to change the compression type and factor.

Availability

Available in Mac OS X v10.0 and later.

See Also- [canBeCompressedUsing:](#) (page 16)**Declared In**

NSBitmapImageRep.h

setPixel:atX:y:

Sets the receiver's pixel at the specified coordinates to the specified raw pixel values.

```
- (void)setPixel:(NSUInteger[])pixelData atX:(NSInteger)x y:(NSInteger)y
```

Parameters

pixelData

An array of integers representing the raw pixel values. The values must be in an order appropriate to the receiver's [bitmapFormat](#) (page 13). Small pixel sample values should be passed as an integer value. Floating point values should be cast `int[]`.

x

The x-axis coordinate of the pixel.

y

The y-axis coordinate of the pixel.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [getPixel:atX:y:](#) (page 20)

Declared In

NSBitmapImageRep.h

setProperty:withValue:

Sets the image's *property* to *value*.

```
- (void)setProperty:(NSString *)property withValue:(id)value
```

Parameters

property

A string constant used as a key for an image property. These properties are described in ["Constants"](#) (page 34).

value

A value specific to *property*. If *value* is `nil`, the value of the property is cleared.

Discussion

The properties can affect how the image is read in and saved to file.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSBitmapImageRep.h

TIFFRepresentation

Returns a TIFF representation of the receiver.

```
- (NSData *)TIFFRepresentation
```

Discussion

This method invokes [TIFFRepresentationUsingCompression:factor:](#) (page 33) using the stored compression type and factor retrieved from the initial image data or changed using [setCompression:factor:](#) (page 31). If the stored compression type isn't supported for writing TIFF data (for example, `NSTIFFCompressionNEXT`), the stored compression is changed to `NSTIFFCompressionNone` before invoking [TIFFRepresentationUsingCompression:factor:](#) (page 33). receiver, using the compression that's returned by [getCompression:factor:](#) (page 19) (if applicable).

If a problem is encountered during generation of the TIFF, `TIFFRepresentation` raises an `NSTIFFException` or an `NSBadBitmapParametersException`.

Availability

Available in Mac OS X v10.0 and later.

See Also

- + [TIFFRepresentationOfImageRepsInArray:](#) (page 11)
- [TIFFRepresentationUsingCompression:factor:](#) (page 33)
- [representationUsingType:properties:](#) (page 29)
- `TIFFRepresentation` (`NSImage`)
- [TIFFRepresentationUsingCompression:factor:](#) (`NSImage`)

Declared In

`NSBitmapImageRep.h`

TIFFRepresentationUsingCompression:factor:

Returns a TIFF representation of the image using the specified compression.

```
- (NSData *)TIFFRepresentationUsingCompression:(NSTIFFCompression)compression
    factor:(float)factor
```

Parameters

compression

An enum constant that represents a TIFF data-compression scheme. Legal values for *compression* can be found in `NSBitmapImageRep.h` and are described in “[Constants](#)” (page 34).

factor

A `float` value that provides a hint for those compression types that implement variable compression ratios.

Currently only JPEG compression uses a compression factor. JPEG compression in TIFF files is not supported, and *factor* is ignored.

Discussion

If the compression type isn't supported for writing TIFF data (for example, `NSTIFFCompressionNEXT`), the stored compression is changed to `NSTIFFCompressionNone` before the TIFF representation is generated.

If a problem is encountered during generation of the TIFF, `TIFFRepresentationUsingCompression:factor:` raises an `NSTIFFException` or an `NSBadBitmapParametersException`.

Availability

Available in Mac OS X v10.0 and later.

See Also

- [canBeCompressedUsing:](#) (page 16)
- + [TIFFRepresentationOfImageRepsInArray:](#) (page 11)
- [TIFFRepresentation](#) (page 32)
- [representationUsingType:properties:](#) (page 29)
- [TIFFRepresentation \(NSImage\)](#)
- [TIFFRepresentationUsingCompression:factor:](#) (NSImage)

Related Sample Code

Quartz Composer Offline Rendering

Declared In

NSBitmapImageRep.h

valueForProperty:

Returns the value for the specified property.

```
- (id)valueForProperty:(NSString *)property
```

Parameters

property

A string constant used as a key for an image property. These properties are described in [“Constants”](#) (page 34).

Return Value

A value specific to *property*, or `nil` if the property is not set for the bitmap.

Discussion

Image properties can affect how an image is read in and saved to file. When retrieving the bitmap image properties defined in [“Bitmap image properties”](#) (page 36), be sure to check the return value of this method for a `nil` value. If a particular value is not set for the image, this method may return `nil`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSBitmapImageRep.h

Constants

NSImageRepLoadStatus

These constants represent the various status values returned by [incrementalLoadFromData:complete:](#) (page 20).

```
enum {
    NSImageRepLoadStatusUnknownType      = -1,
    NSImageRepLoadStatusReadingHeader    = -2,
    NSImageRepLoadStatusWillNeedAllData  = -3,
    NSImageRepLoadStatusInvalidData      = -4,
    NSImageRepLoadStatusUnexpectedEOF    = -5,
    NSImageRepLoadStatusCompleted        = -6
};
typedef NSInteger NSImageRepLoadStatus;
```

Constants

`NSImageRepLoadStatusUnknownType`

Not enough data to determine image format. You should continue to provide more data.

Available in Mac OS X v10.2 and later.

Declared in `NSBitmapImageRep.h`.

`NSImageRepLoadStatusReadingHeader`

The image format is known, but not enough data has been read to determine the size, depth, etc., of the image. You should continue to provide more data.

Available in Mac OS X v10.2 and later.

Declared in `NSBitmapImageRep.h`.

`NSImageRepLoadStatusWillNeedAllData`

Incremental loading cannot be supported. Until you call [incrementalLoadFromData:complete:](#) (page 20) with YES, this status will be returned. You can continue to call the method but no decompression will take place. Once you do call the method with YES, then the image will be decompressed and one of the final three status messages will be returned.

Available in Mac OS X v10.2 and later.

Declared in `NSBitmapImageRep.h`.

`NSImageRepLoadStatusInvalidData`

An error occurred during image decompression. The image contains the portions of the data that have already been successfully decompressed, if any

Available in Mac OS X v10.2 and later.

Declared in `NSBitmapImageRep.h`.

`NSImageRepLoadStatusUnexpectedEOF`

[incrementalLoadFromData:complete:](#) (page 20) was called with YES, but not enough data was available for decompression. The image contains the portions of the data that have already been successfully decompressed, if any.

Available in Mac OS X v10.2 and later.

Declared in `NSBitmapImageRep.h`.

`NSImageRepLoadStatusCompleted`

Enough data has been provided to successfully decompress the image (regardless of the complete: flag).

Available in Mac OS X v10.2 and later.

Declared in `NSBitmapImageRep.h`.

Bitmap image properties

These constants identify properties that are used by [representationOfImageRepsInArray:usingType:properties:](#) (page 11), [representationUsingType:properties:](#) (page 29), [setPixel:atX:y:](#) (page 32), and [valueForProperty:](#) (page 34).

```
NSString *NSImageCompressionMethod;
NSString *NSImageCompressionFactor;
NSString *NSImageDitherTransparency;
NSString *NSImageRGBColorTable;
NSString *NSImageInterlaced;
NSString *NSImageColorSyncProfileData;
NSString *NSImageFrameCount;
NSString *NSImageCurrentFrame;
NSString *NSImageCurrentFrameDuration;
NSString *NSImageLoopCount;
NSString *NSImageGamma;
NSString *NSImageProgressive;
NSString *NSImageEXIFData;
NSString* NSImageFallbackBackgroundColor
```

Constants

`NSImageColorSyncProfileData`

Identifies an `NSData` object containing the ColorSync profile data.

It can be used for TIFF, JPEG, GIF, and PNG files. This value is set when reading in and used when writing out image data. You can get the profile data for a particular color space from the corresponding `NSColorSpace` object or from the ColorSync Manager.

Available in Mac OS X v10.0 and later.

Declared in `NSBitmapImageRep.h`.

`NSImageCompressionFactor`

Identifies an `NSNumber` object containing the compression factor of the image.

Used only for JPEG files. JPEG compression in TIFF files is not supported, and the factor is ignored. The value is a float between 0.0 and 1.0, with 1.0 resulting in no compression and 0.0 resulting in the maximum compression possible. It's set when reading in and used when writing out the image.

Available in Mac OS X v10.0 and later.

Declared in `NSBitmapImageRep.h`.

`NSImageCompressionMethod`

Identifies an `NSNumber` object identifying the compression method of the image.

Used only for TIFF files. The value corresponds to one of the `NSTIFFCompression` constants, described below. It's set when reading in and used when writing out.

Available in Mac OS X v10.0 and later.

Declared in `NSBitmapImageRep.h`.

`NSImageDitherTransparency`

Identifies an `NSNumber` object containing a boolean that indicates whether the image is dithered.

Used only when writing GIF files.

Available in Mac OS X v10.0 and later.

Declared in `NSBitmapImageRep.h`.

NSImageInterlaced

Identifies an `NSNumber` object containing a Boolean value that indicates whether the image is interlaced.

Used only when writing out PNG files.

Available in Mac OS X v10.0 and later.

Declared in `NSBitmapImageRep.h`.

NSImageRGBColorTable

Identifies an `NSData` object containing the RGB color table.

Used only for GIF files. It's stored as packed RGB. It's set when reading in and used when writing out.

Available in Mac OS X v10.0 and later.

Declared in `NSBitmapImageRep.h`.

NSImageEXIFData

Identifies an `NSDictionary` object containing the EXIF data for the image.

This property is used only when reading or writing JPEG files. The dictionary contains the EXIF keys and values. The standard dictionary keys (that is, those that are not specific to camera vendors) are identical to those for `kCGImagePropertyExifDictionary` declared in the `CGImageSource` API. See `kCGImagePropertyExifDictionary` Keys for details.

Available in Mac OS X v10.4 and later.

Declared in `NSBitmapImageRep.h`.

NSImageFallbackBackgroundColor

Specifies the background color to use when writing to an image format (such as JPEG) that doesn't support alpha. The color's alpha value is ignored. The default background color, when this property is not specified, is white. The value of the property should be an `NSColor` object. This constant corresponds to the `kCGImageDestinationBackgroundColor` constant in Quartz.

Available in Mac OS X v10.5 and later.

Declared in `NSBitmapImageRep.h`.

NSImageFrameCount

Identifies an `NSNumber` object containing the number of frames in an animated GIF file.

This value is used when reading in data.

Available in Mac OS X v10.2 and later.

Declared in `NSBitmapImageRep.h`.

NSImageGamma

Identifies an `NSNumber` object containing the gamma value for the image.

Used only for PNG files. The gamma value is a floating-point number between 0.0 and 1.0, with 0.0 being black and 1.0 being the maximum color. It's set when reading in and used when writing out.

Available in Mac OS X v10.4 and later.

Declared in `NSBitmapImageRep.h`.

NSImageCurrentFrame

Identifies an `NSNumber` object containing the current frame for an animated GIF file.

The first frame is 0.

Available in Mac OS X v10.2 and later.

Declared in `NSBitmapImageRep.h`.

`NSImageCurrentFrameDuration`

Identifies an `NSNumber` object containing the duration (in seconds) of the current frame for an animated GIF image.

The frame duration can be a floating-point value. It is used when reading in, but not when writing out.

Available in Mac OS X v10.2 and later.

Declared in `NSBitmapImageRep.h`.

`NSImageProgressive`

Identifies an `NSNumber` object containing a boolean that indicates whether the image uses progressive encoding.

Used only for JPEG files. It's set when reading in and used when writing out.

Available in Mac OS X v10.4 and later.

Declared in `NSBitmapImageRep.h`.

`NSImageLoopCount`

Identifies an `NSNumber` object containing the number of loops to make when animating a GIF image.

A value of 0 indicates the animation should loop indefinitely. Values should be specified as integer numbers. It is used when reading in but not when writing out the image.

Available in Mac OS X v10.3 and later.

Declared in `NSBitmapImageRep.h`.

Discussion

When using the `valueForProperty:` method to retrieve the the value for any of these keys, be sure to check that the returned value is non-`nil` before you attempt to use it. A bitmap image representation may return `nil` for any values that have not yet been set.

NSBitmapImageFileType

The following file type constants are provided as a convenience by `NSBitmapImageRep`:

```
enum {
    NSTIFFFileType,
    NSBMPFileType,
    NSGIFFileType,
    NSJPEGFileType,
    NSPNGFileType,
    NSJPEG2000FileType
};
typedef NSUInteger NSBitmapImageFileType;
```

Constants`NSTIFFFileType`

Tagged Image File Format (TIFF)

Available in Mac OS X v10.0 and later.

Declared in `NSBitmapImageRep.h`.

`NSBMPFileType`

Windows bitmap image (BMP) format

Available in Mac OS X v10.0 and later.

Declared in `NSBitmapImageRep.h`.

NSGIFFileType

Graphics Image Format (GIF), originally created by CompuServe for online downloads

Available in Mac OS X v10.0 and later.

Declared in NSBitmapImageRep.h.

NSJPEGFileType

JPEG format

Available in Mac OS X v10.0 and later.

Declared in NSBitmapImageRep.h.

NSPNGFileType

Portable Network Graphics (PNG) format

Available in Mac OS X v10.0 and later.

Declared in NSBitmapImageRep.h.

NSJPEG2000FileType

JPEG 2000 file format.

Available in Mac OS X v10.4 and later.

Declared in NSBitmapImageRep.h.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSBitmapImageRep.h

NSTIFFCompression

These constants represent the various TIFF data-compression schemes supported by NSBitmapImageRep.

```
enum {
    NSTIFFCompressionNone = 1,
    NSTIFFCompressionCCITTFAX3 = 3,
    NSTIFFCompressionCCITTFAX4 = 4,
    NSTIFFCompressionLZW = 5,
    NSTIFFCompressionJPEG = 6,
    NSTIFFCompressionNEXT = 32766,
    NSTIFFCompressionPackBits = 32773,
    NSTIFFCompressionOldJPEG = 32865
};
typedef NSUInteger NSTIFFCompression;
```

Constants

NSTIFFCompressionNone

No compression.

Available in Mac OS X v10.0 and later.

Declared in NSBitmapImageRep.h.

NSTIFFCompressionCCITTFAX3

CCITT Fax Group 3 compression.

Used for 1-bit fax images sent over telephone lines.

Available in Mac OS X v10.0 and later.

Declared in NSBitmapImageRep.h.

`NSTIFFCompressionCCITTFAX4`

CCITT Fax Group 4 compression.

Used for 1-bit fax images sent over ISDN lines.

Available in Mac OS X v10.0 and later.

Declared in `NSBitmapImageRep.h`.

`NSTIFFCompressionLZW`

LZW compression.

Available in Mac OS X v10.0 and later.

Declared in `NSBitmapImageRep.h`.

`NSTIFFCompressionJPEG`

JPEG compression. No longer supported for input or output.

Available in Mac OS X v10.0 and later.

Declared in `NSBitmapImageRep.h`.

`NSTIFFCompressionNEXT`

NeXT compressed. Supported for input only.

Available in Mac OS X v10.0 and later.

Declared in `NSBitmapImageRep.h`.

`NSTIFFCompressionPackBits`

PackBits compression.

Available in Mac OS X v10.0 and later.

Declared in `NSBitmapImageRep.h`.

`NSTIFFCompressionOldJPEG`

Old JPEG compression. No longer supported for input or output.

Available in Mac OS X v10.0 and later.

Declared in `NSBitmapImageRep.h`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSBitmapImageRep.h`

NSBitmapFormat

These constants represent the various bitmap component formats supported by `NSBitmapImageRep`. These values are combined using the C bitwise OR operator and passed to

`initWithBitmapDataPlanes:pixelWide:pixelHigh:bitsPerSample:samplesPerPixel:hasAlpha:isPlanar:colorSpaceName:bitmapFormat:bytesPerRow:bitsPerPixel:` (page 22) as the bitmap format and are returned by `bitmapFormat` (page 13).

```
enum {  
    NSAlphaFirstBitmapFormat          = 1 << 0,  
    NSAlphaNonpremultipliedBitmapFormat = 1 << 1,  
    NSFloatingPointSamplesBitmapFormat = 1 << 2  
};  
typedef NSUInteger NSBitmapFormat;
```

Constants

NSAlphaFirstBitmapFormat

If 0, alpha values are the last component.

For example, CMYKA and RGBA.

Available in Mac OS X v10.4 and later.

Declared in NSBitmapImageRep.h.

NSAlphaNonpremultipliedBitmapFormat

If 0, alpha values are premultiplied.

Available in Mac OS X v10.4 and later.

Declared in NSBitmapImageRep.h.

NSFloatingPointSamplesBitmapFormat

If 0, samples are integer values.

Available in Mac OS X v10.4 and later.

Declared in NSBitmapImageRep.h.

Availability

Available in Mac OS X v10.4 and later.

Declared In

NSBitmapImageRep.h

Document Revision History

This table describes the changes to *NSBitmapImageRep Class Reference*.

Date	Notes
2009-06-28	Updated for Mac OS X v10.6. New colorspace management methods.
2009-01-06	Updated the descriptions of the initWithCGImage: and initWithCIImage: methods.
2008-10-15	Fixed a typo in the initWithCIImage: method.
2007-02-28	Corrected description of the NSImageCompressionFactor constant. Updated for Mac OS X version 10.5.
2006-05-23	First publication of this content as a separate document.

REVISION HISTORY

Document Revision History

Index

B

Bitmap image properties [36](#)
`bitmapData` **instance method** [12](#)
`bitmapFormat` **instance method** [13](#)
`bitmapImageRepByConvertingToColorSpace:`
 `renderingIntent`: **instance method** [13](#)
`bitmapImageRepByRetaggingWithColorSpace:`
 instance method [14](#)
`bitsPerPixel` **instance method** [14](#)
`bytesPerPlane` **instance method** [15](#)
`bytesPerRow` **instance method** [15](#)

C

`canBeCompressedUsing`: **instance method** [16](#)
`CGImage` **instance method** [16](#)
`colorAtX:y`: **instance method** [17](#)
`colorizeByMappingGray:toColor:blackMapping:`
 `whiteMapping`: **instance method** [17](#)
`colorSpace` **instance method** [18](#)

G

`getBitmapDataPlanes`: **instance method** [18](#)
`getCompression:factor`: **instance method** [19](#)
`getPixel:atX:y`: **instance method** [20](#)
`getTIFFCompressionTypes:count`: **class method** [9](#)

I

`imageRepsWithData`: **class method** [9](#)
`imageRepWithData`: **class method** [10](#)
`incrementalLoadFromData:complete`: **instance method** [20](#)
`initWithIncrementalLoad` **instance method** [21](#)

`initWithBitmapDataPlanes:pixelsWide:pixelsHigh:`
 `bitsPerSample:samplesPerPixel:hasAlpha:isPlanar:`
 `colorSpaceName:bitmapFormat:bytesPerRow:`
 `bitsPerPixel`: **instance method** [22](#)
`initWithBitmapDataPlanes:pixelsWide:pixelsHigh:`
 `bitsPerSample:samplesPerPixel:hasAlpha:isPlanar:`
 `colorSpaceName:bytesPerRow:bitsPerPixel:`
 instance method [24](#)
`initWithCGImage`: **instance method** [26](#)
`initWithCIImage`: **instance method** [27](#)
`initWithData`: **instance method** [28](#)
`initWithFocusedViewRect`: **instance method** [28](#)
`isPlanar` **instance method** [29](#)

L

`localizedNameForTIFFCompressionType`: **class method** [10](#)

N

`NSAlphaFirstBitmapFormat` **constant** [41](#)
`NSAlphaNonpremultipliedBitmapFormat` **constant** [41](#)
`NSBitmapFormat` **data type** [40](#)
`NSBitmapImageFileType` **data type** [38](#)
`NSBMPFileType` **constant** [38](#)
`NSFloatingPointSamplesBitmapFormat` **constant** [41](#)
`NSGIFFileType` **constant** [39](#)
`NSImageColorSyncProfileData` **constant** [36](#)
`NSImageCompressionFactor` **constant** [36](#)
`NSImageCompressionMethod` **constant** [36](#)
`NSImageCurrentFrame` **constant** [37](#)
`NSImageCurrentFrameDuration` **constant** [38](#)
`NSImageDitherTransparency` **constant** [36](#)
`NSImageEXIFData` **constant** [37](#)
`NSImageFallbackBackgroundColor` **constant** [37](#)
`NSImageFrameCount` **constant** [37](#)
`NSImageGamma` **constant** [37](#)

NSImageInterlaced **constant** [37](#)
 NSImageLoopCount **constant** [38](#)
 NSImageProgressive **constant** [38](#)
NSImageRepLoadStatus [34](#)
 NSImageRepLoadStatusCompleted **constant** [35](#)
 NSImageRepLoadStatusInvalidData **constant** [35](#)
 NSImageRepLoadStatusReadingHeader **constant** [35](#)
 NSImageRepLoadStatusUnexpectedEOF **constant** [35](#)
 NSImageRepLoadStatusUnknownType **constant** [35](#)
 NSImageRepLoadStatusWillNeedAllData **constant**
[35](#)
 NSImageRGBColorTable **constant** [37](#)
 NSJPEG2000FileType **constant** [39](#)
 NSJPEGFileType **constant** [39](#)
 NSPNGFileType **constant** [39](#)
 NSTIFFCompression **data type** [39](#)
 NSTIFFCompressionCCITTFAX3 **constant** [39](#)
 NSTIFFCompressionCCITTFAX4 **constant** [40](#)
 NSTIFFCompressionJPEG **constant** [40](#)
 NSTIFFCompressionLZW **constant** [40](#)
 NSTIFFCompressionNEXT **constant** [40](#)
 NSTIFFCompressionNone **constant** [39](#)
 NSTIFFCompressionOldJPEG **constant** [40](#)
 NSTIFFCompressionPackBits **constant** [40](#)
 NSTIFFFileType **constant** [38](#)
 numberOfPlanes **instance method** [29](#)

R

representationOfImageRepsInArray:usingType:
 properties: **class method** [11](#)
 representationUsingType:properties: **instance
 method** [29](#)

S

samplesPerPixel **instance method** [30](#)
 setColor:atX:y: **instance method** [30](#)
 setCompression:factor: **instance method** [31](#)
 setPixel:atX:y: **instance method** [32](#)
 setProperty:withValue: **instance method** [32](#)

T

TIFFRepresentation **instance method** [32](#)
 TIFFRepresentationOfImageRepsInArray: **class
 method** [11](#)
 TIFFRepresentationOfImageRepsInArray:
 usingCompression:factor: **class method** [12](#)

TIFFRepresentationUsingCompression:factor:
instance method [33](#)

V

valueForProperty: **instance method** [34](#)