NSBitmapImageRep

Inherits From:	NSImageRep : NSObject
Conforms To:	NSCoding (from NSImageRep) NSCopying (from NSImageRep) NSObject (from NSObject)
Declared In:	AppKit/NSImage.h

Class Description

An NSBitmapImageRep is an object that can render an image from bitmap data. The data can be in Tag Image File Format (TIFF), Windows bitmap format (BMP), or it can be raw image data. If it's raw data, the object must be informed about the structure of the image—its size, the number of color components, the number of bits per sample, and so on—when it's first initialized. If it's TIFF or BMP data, the object can get this information from the various fields included with the data.

Although NSBitmapImageReps are often used indirectly, through instances of the NSImage class, they can also be used directly—for example to manipulate the bits of an image as you might need to do in a paint program.

Setting Up an NSBitmapImageRep

You pass bitmap data for an image to a new NSBitmapImageRep when you first initialize it. You can also create an NSBitmapImageRep from bitmap data that's read from a specified rectangle of a focused NSView.

Although the NSBitmapImageRep class inherits NSImageRep methods that set image attributes, these methods shouldn't be used. Instead, you should either allow the object to find out about the image from the fields included with the bitmap data, or use methods defined in this class to supply this information when the object is initialized.

TIFF Compression

TIFF data can be read and rendered after it has been compressed using any one of the four schemes briefly described below:

LZW	Compresses and decompresses without information loss, achieving compression ratios up to 5:1. It may be somewhat slower to compress and decompress than the PackBits scheme.
PackBits	Compresses and decompresses without information loss, but may not achieve the same compression ratios as LZW.
JPEG	Compresses and decompresses with some information loss, but can achieve compression ratios anywhere from 10:1 to 100:1. The ratio is determined by a user-settable factor ranging from 1.0 to 255.0, with higher factors yielding greater compression. More information is lost with greater compression, but 15:1 compression is safe for publication quality. Some images can be compressed even more. JPEG compression can be used only for images that specify at least 4 bits per sample.
CCITTFAX	Compresses and decompresses 1 bit gray-scale images using international fax compression standards CCITT3 and CCITT4.

An NSBitmapImageRep can also produce compressed TIFF data for its image using any of these schemes.

Method Types

Creating an NSBitmapImageRep	+ imageRepsWithData: + imageRepWithData: – initWithBitmapDataPlanes:pixelsWide:pixelsHigh:bitsPerSample:samplesPerPixel:hasAlpha:isPlanar:colorSpaceName:bytesPerRow:bitsPerPixel: – initWithBitmapHandle: – initWithData: – initWithFocusedViewRect: – initWithIconHandle:
Getting information about the image	– bitsPerPixel – bytesPerPlane – bytesPerRow – isPlanar – numberOfPlanes – samplesPerPixel

Getting image data	– bitmapData – getBitmapDataPlanes:
Producing a TIFF representation of the image	+ TIFFRepresentationOfImageRepsInArray: + TIFFRepresentationOfImageRepsInArray:usingCompression:factor: – TIFFRepresentation – TIFFRepresentationUsingCompression:factor:
Setting and checking compression types	+ getTIFFCompressionTypes:count: + localizedNameForTIFFCompressionType: – canBeCompressedUsing: – getCompression:factor: – setCompression:factor:

Class Methods

getTIFFCompressionTypes:count:

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

Returns, by reference, an array of NSTIFFCompressions representing all available compression types that can be used when writing a TIFF image. The number of elements in list is represented by *numTypes*. *list* belongs to the NSBitmapImageRep class; it shouldn't be freed or altered.

The following compression types are supported:

Constant	Value	Usage
NSTIFFCompressionNone	1	
NSTIFFCompressionCCITTFAX3	3	1 bps images only
NSTIFFCompressionCCITTFAX4	4	1 bps images only
NSTIFFCompressionLZW	5	
NSTIFFCompressionJPEG	6	
NSTIFFCompressionNEXT	32766	Input only
NSTIFFCompressionPackBits	32773	
NSTIFFCompressionOldJPEG	32865	Input only

Note that not all compression types can be used for all images: NSTIFFCompressionNEXT can be used only to retrieve image data. Because future releases of OpenStep 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.

See also: **+ localizedNameForTIFFCompressionType:**, **– canBeCompressedUsing:**

imageRepsWithData:

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

Creates and returns an array of initialized NSBitmapImageRep objects corresponding to the images in *bitmapData*. If NSBitmapImageRep is unable to interpret *bitmapData*, the returned array is empty. *bitmapData* can contain data in any supported bitmap format.

imageRepWithData:

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

Creates and returns an initialized NSBitmapImageRep corresponding to the first image in *bitmapData*, or **nil** if NSBitmapImageRep is unable to interpret *bitmapData*. *bitmapData* can contain data in any supported bitmap format.

localizedNameForTIFFCompressionType:

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

Returns an autoreleased string containing the localized name for the compression type represented by *compression*, or **nil** if *compression* is unrecognized. Compression types are listed in the **getTIFFCompressionTypes:count:** class method description. When implementing a user interface for selecting TIFF compression types, use **getTIFFCompressionTypes:count:** to get the list of supported compression types, then use this method to get the localized names for each compression type.

See also: + **getTIFFCompressionTypes:count:**

TIFFRepresentationOfImageRepsInArray:

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

Returns a TIFF representation of the images in *array*, using the compression that's returned by **getCompression:factor:** (if applicable).

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

See also: – **TIFFRepresentation**

TIFFRepresentationOfImageRepsInArray:usingCompression:factor:

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

Returns a TIFF representation of the images in array, which are compressed using the specified compression type and factor. Legal values for *compression* can be found in **NSBitmapImageRep.h**, and are described in “Tiff Compression” in NSBitmapImageRep’s class description. *factor* provides a hint for those compression types that implement variable compression ratios; currently only JPEG compression uses a compression factor. If your compression type doesn’t implement variable compression ratios, or if it does and you don’t want the image to be compressed, specify a compression factor of 0.0.

If the specified compression isn’t applicable, no compression is used. If a problem is encountered during generation of the TIFF, **TIFFRepresentationOfImageRepsInArray:usingCompression:factor:** raises an NSTIFFException or an NSBadBitmapParametersException.

See also: – **TIFFRepresentationUsingCompression:factor:**

Instance Methods

bitmapData

– (unsigned char *)**bitmapData**

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

See also: – **getBitmapDataPlanes:**

bitsPerPixel

– (int)**bitsPerPixel**

Returns the number of bits allocated for each pixel in each plane of data. This 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 the **initWithBitmapDataPlanes:pixelsWide:pixelsHigh:...** method) in case extra memory is allocated for each pixel. This may be the case, for example, if pixel data is aligned on byte boundaries.

bytesPerPlane

– (int)**bytesPerPlane**

Returns the number of bytes in each plane or channel of data. This is calculated from the number of bytes per row and the height of the image.

See also: – **bytesPerRow**

bytesPerRow

– (int)**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. If not explicitly set to another value (in the **initWithBitmapDataPlanes:pixelsWide:pixelsHigh:...** method), this 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.

See also: – **bytesPerPlane**

canBeCompressedUsing:

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

Tests whether the receiver can be compressed by compression type. Legal values for *compression* can be found in **NSBitmapImageRep.h**, and are described in “Tiff Compression” in the class description. This method returns YES if the receiver’s data matches compression; for example, if compression is NSTIFFCompressionCCITTFAX3, then the data must be one bit-per-sample and one sample-per-pixel. It returns NO if the data doesn’t match compression or if compression is unsupported.

See also: + **getTIFFCompressionTypes:count:**

getBitmapDataPlanes:

– (void)**getBitmapDataPlanes:**(unsigned char **)*data*

Provides access to bitmap data for the image separated into planes. *data* should be an 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.

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.

See also: – **data**, – **isPlanar**

getCompression:factor:

– (void)**getCompression:**(NSTIFFCompression *)*compression* **factor:**(float *)*factor*

Returns by reference the receiver’s compression type and compression factor. Use this method to get information on the compression type for the source image data. *compression* represents the compression type used on the data, and corresponds to one of the values returned by the class method getTIFFCompressionTypes:count:. factor is a 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 255.0, with 0.0 representing minimal compression.

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

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

Initializes the receiver, a newly allocated NSBitmapImageRep object, so that it can render the image specified in *planes* and described by the other arguments. If the object can’t be initialized, this method frees it and returns **nil**. Otherwise, it returns the object (**self**).

planes is 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 the data is in meshed configuration (*isPlanar* is NO), only the first buffer is read.

If *planes* is NULL or if it’s 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 **getBitmapDataPlanes:** or **bitmapData** 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 object will only reference the image data; it won’t copy it. The buffers won’t be freed when the object is freed.

Each of the other arguments (besides *planes*) informs the NSBitmapImageRep object about the image. They’re explained below:

- *width* and *height* specify the size of the image in pixels. The size in each direction must be greater than 0.
- *bps* (bits per sample) is 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.
- *spp* (samples per pixel) is the number of data components. It 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 gray-scale image that lacks a coverage component would have an *spp* of 1.
- *alpha* should be YES if one of the components counted in the number of samples per pixel (*spp*) is a coverage component, and NO if there is no coverage component.

- *isPlanar* should be 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”).

  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* indicates how data values are to be interpreted. It should be one of the following enumerated values:

  NSCalibratedWhiteColorSpace

  NSCalibratedBlackColorSpace

  NSCalibratedRGBColorSpace

  NSDeviceWhiteColorSpace

  NSDeviceBlackColorSpace

  NSDeviceRGBColorSpace

  NSDeviceCMYKColorSpace

  NSNamedColorSpace

  NSCustomColorSpace

- *rowBytes* is 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 *rowBytes* is 0, the NSBitmapImageRep assumes that there’s no empty space at the end of a row.

- *pixelBits* informs the 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 12 *bps*, *pixelBits* may be 16; for RGB images with 24 *bps*, *pixelBits* may be 32. The legal values for *pixelBits* are system dependent.

  If *pixelBits* is 0, the object will interpret the number of bits per pixel to be the expected value, without any meaningless bits.

initWithBitmapHandle:

– (id)**initWithBitmapHandle:**(void *)*bitmap*

On Microsoft Windows platforms, **initWithBitmapHandle:** initializes the receiver, a newly allocated NSBitmapImageRep instance, with the contents of the Windows bitmap indicated by *bitmap*. If **initWithBitmapHandle:** is able to create one or more image representations, it returns **self**. Otherwise, the receiver is freed and **nil** is returned.

initWithData:

– (id)**initWithData:**(NSData *)*bitmapData*

Initializes a newly allocated NSBitmapImageRep from the data found in *bitmapData*. 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*.

initWithData: returns an initialized NSBitmapImageRep if the initialization was successful, or **nil** if it was unable to interpret the contents of *bitmapData*.

initWithFocusedViewRect:

– (id)**initWithFocusedViewRect:**(NSRect)*rect*

Initializes the receiver, a newly allocated NSBitmapImageRep object, with bitmap data read from a rendered image. The image that's read is located in the current window and is bounded by the *rect* rectangle as specified in the current coordinate system.

This method uses PostScript 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.

If for any reason the new object can't be initialized, this method frees it and returns **nil**. Otherwise, it returns the initialized object (**self**).

initWithIconHandle:

– (id)**initWithIconHandle:**(void *)*icon*

On Microsoft Windows platforms, **initWithIconHandle:** initializes the receiver, a newly allocated NSBitmapImageRep instance, with the contents of the Windows icon indicated by *icon*. If **initWithIconHandle:** is able to create one or more image representations, it returns **self**. Otherwise, the receiver is freed and **nil** is returned.

isPlanar

– (BOOL)**isPlanar**

Returns YES if image data is segregated into a separate plane for each color and coverage component (planar configuration), and NO if the data is integrated into a single plane (meshed configuration).

See also: – **samplesPerPixel**

numberOfPlanes

– (int)**numberOfPlanes**

Returns the number of separate planes that image data is organized into. This is the number of samples per pixel if the data has a separate plane for each component (**isPlanar** returns YES) and 1 if the data is meshed (**isPlanar** returns NO).

See also: – **samplesPerPixel**, – **hasAlpha** (NSImageRep), – **bitsPerSample** (NSImageRep)

samplesPerPixel

– (int)**samplesPerPixel**

Returns the number of components in the data. It includes both color components and the coverage component, if present.

See also: – **hasAlpha** (NSImageRep), – **bitsPerSample** (NSImageRep)

setCompression:factor:

– (void)**setCompression:**(NSTIFFCompression)*compression* **factor:**(float)*factor*

Sets the receiver's compression type and compression factor. *compression* is one of the supported compression types listed in the **getTiffCompressionTypes:count:** class method description. *factor* is a 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 255.0, with 0.0 representing minimal compression.

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

See also: – **canBeCompressedUsing:**

TIFFRepresentation

– (NSData *)**TIFFRepresentation**

Returns a TIFF representation of the image, using the compression that's returned by **getCompression:factor:** (if applicable). This method invokes **TIFFRepresentationUsingCompression:factor:** using the stored compression type and factor retrieved from the initial image data or changed using **setCompression:factor:**. If the stored compression type isn't supported for writing TIFF data (for example, NSTIFFCompressionNEXT), the stored compression is changed to NSTIFFCompressionNone and the compression factor to 0.0 before invoking **TIFFRepresentationUsingCompression:factor:**.

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

See also: + **TIFFRepresentationOfImageRepsInArray:**

TIFFRepresentationUsingCompression:factor:

– (NSData *)**TIFFRepresentationUsingCompression:**(NSTIFFCompression)*comp*
  factor:(float)*factor*

Returns a TIFF representation of the image, using the specified compression and factor. If the stored compression type isn't supported for writing TIFF data (for example, NSTIFFCompressionNEXT), the stored compression is changed to NSTIFFCompressionNone and the compression factor to 0.0 before the TIFF representation is generated.

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

See also: **– canBeCompressedUsing:**, **+ TIFFRepresentationOfImageRepsInArray:**