NSImage

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

Class Description

An NSImage object contains an image that can be composited anywhere without first being drawn in any particular view. It manages the image by:

- Reading image data from the application bundle, from an NSPasteboard, or from an NSData object.
- Keeping multiple representations of the same image.
- Choosing the representation that's appropriate for a particular data type.
- Choosing the representation that's appropriate for any given display device.
- Caching the representations it uses by rendering them in off-screen windows.
- Optionally retaining the data used to draw the representations, so that they can be reproduced when needed.
- Compositing the image from the off-screen cache to where it's needed on-screen.
- Reproducing the image for the printer so that it matches what's displayed on-screen, yet is the best representation possible for the printed page.
- Automatically using any filtering services installed by the user to convert image data from unsupported formats to supported formats.

Defining an Image

An image can be created from various types of data:

- Encapsulated PostScript code (EPS)
- Bitmap data in Tag Image File Format (TIFF)
- Bitmap data in Windows Bitmap format (BMP)
- Untagged (raw) bitmap data

- Other image data supported by an NSImageRep subclass registered with the NSImage class
- Data that can be filtered to a supported type by a user-installed filter service

If data is placed in a file (for example, in an application bundle), the NSImage object can access the data whenever it's needed to create the image. If data is read from an NSData object, the NSImage object may need to store the data itself.

Images can also be defined by the program, in two ways:

- By drawing the image in an off-screen window maintained by the NSImage object. In this case, the NSImage maintains only the cached image.
- By defining a method that can be used to draw the image when needed. This allows the NSImage to delegate responsibility for producing the image to some other object.

Image Representations

An NSImage object can keep more than one representation of an image. Multiple representations permit the image to be customized for the display device. For example, different hand-tuned TIFF images can be provided for monochrome and color screens, and an EPS representation or a custom method might be used for printing. All representations are versions of the same image.

An NSImage returns an NSArray of its representations in response to a **representations** message. Each representation is a kind of NSImageRep object:

NSEPSImageRep	An image that can be recreated from EPS data that's either stored by the object or at a known location in the file system.
NSBitmapImageRep	An image that can be recreated from bitmap or TIFF data.
NSCustomImageRep	An image that can be redrawn by a method defined in the application.
NSCachedImageRep	An image that has been rendered in an off-screen cache from data or instructions that are no longer available. The image in the cache provides the only data from which the image can be reproduced.

You can define other NSImageRep subclasses for objects that render images from other types of source data. To make these new subclasses available to an NSImage object, they need to be added to the NSImageRep class registry by invoking the **registerImageRepClass:** class method. NSImage determines the data types that each subclass can support by invoking its **imageUnfilteredFileTypes** and **imageUnfilteredPasteboardTypes** methods.

Choosing Representations

The NSImage object will choose the representation that best matches the rendering device. By default, the choice is made according to the following set of ordered rules. Each rule is applied in turn until the choice of representation is narrowed to one.

1. Choose a color representation for a color device, and a gray-scale representation for a monochrome device.
2. Choose a representation with a resolution that matches the resolution of the device, or if no representation matches, choose the one with the highest resolution.

   By default, any image representation with a resolution that's an integer multiple of the device resolution is considered to match. If more than one representation matches, the NSImage will choose the one that's closest to the device resolution. However, you can force resolution matches to be exact by passing NO to the **setMatchesOnMultipleResolution:** method.

   Rule 2 prefers TIFF and bitmap representations, which have a defined resolution, over EPS representations, which don't. However, you can use the **setUsesEPSOnResolutionMismatch:** method to have the NSImage choose an EPS representation in case a resolution match isn't possible.
3. If all else fails, choose the representation with a specified bits per sample that matches the depth of the device. If no representation matches, choose the one with the highest bits per sample.

By passing NO to the **setPrefersColorMatch:** method, you can have the NSImage try for a resolution match before a color match. This essentially inverts the first and second rules above.

If these rules fail to narrow the choice to a single representation—for example, if the NSImage has two color TIFF representations with the same resolution and depth—the one that will be chosen is system dependent.

Caching Representations

When first asked to composite the image, the NSImage object chooses the representation that's best for the destination display device, as outlined above. It renders the representation in an off-screen window on the same device, then composites it from this cache to the desired location. Subsequent requests to composite the image use the same cache. Representations aren't cached until they're needed for compositing.

When printing, the NSImage tries not to use the cached image. Instead, it attempts to render on the printer—using the appropriate image data, or a delegated method—the best version of the image that it can. Only as a last resort will it image the cached bitmap.

Image Size

Before an NSImage can be used, the size of the image must be set, in units of the base coordinate system. If a representation is smaller or larger than the specified size, it can be scaled to fit.

If the size of the image hasn't already been set when the NSImage is provided with a representation, the size will be set from the data. The bounding box is used to determine the size of an NSEPSImageRep. The TIFF fields "ImageLength" and "ImageWidth" are used to determine the size of an NSBitmapImageRep.

Coordinate Systems

Images have the horizontal orientation of the base coordinate system; they can't be rotated or horizontally flipped. When composited, an image maintains this orientation, no matter what coordinate system it's composited to. (The destination coordinate system is used only to determine the location of a composited image, not its size or orientation.) Images can be flipped in the vertical direction by using **setFlipped:**.

It's possible to refer to portions of an image when compositing by specifying a rectangle in the image's coordinate system, which is identical to the base coordinate system, except that the origin is at the lower left corner of the image.

Named Images

An NSImage object can be identified either by its **id** or by a name. Assigning an NSImage a name adds it to a table kept by the class object; each name in the database identifies one and only one instance of the class. When you ask for an NSImage object by name (with the **imageNamed:** method), the class object returns the one from its database, which also includes all the system bitmaps provided by the Application Kit. If there's no object in the database for the specified name, the class object tries to create one by checking for a system bitmap of the same name, checking the name of the application's own image, and then checking for the image in the application's main bundle.

If a file matches the name, an NSImage is created from the data stored there. You can therefore create NSImage objects simply by including EPS, BMP, or TIFF data for them within the executable file, or in files inside the application's file package.

Image Filtering Services

NSImage is designed to automatically take advantage of user-installed filter services for converting unsupported image file types to supported image file types. The class method **imageFileTypes** returns an array of all file types from which NSImage can create an instance of itself. This list includes all file types supported by registered subclasses of NSImageRep, and those types that can be converted to supported file types through a user-installed filter service.

Adopted Protocols

NSCoding	– encodeWithCoder: – initWithCoder:
NSCopying	– copyWithZone:

Method Types

Initializing a new NSImage instance
- initByReferencingFile:
- initWithBitmapHandle:
- initWithContentsOfFile:
- initWithData:
- initWithIconHandle:
- initWithPasteboard:
- initWithSize:

Setting the size of the image
- setSize:
- size

Referring to images by name
- + imageNamed:
- setName:
- name

Specifying the image
- addRepresentation:
- addRepresentations:
- lockFocus
- lockFocusOnRepresentation:
- unlockFocus

Using the image
- compositeToPoint:operation:
- compositeToPoint:fromRect:operation:
- dissolveToPoint:fraction:
- dissolveToPoint:fromRect:fraction:

Choosing which image representation to use
- setPrefersColorMatch:
- prefersColorMatch
- setUsesEPSOnResolutionMismatch:
- usesEPSOnResolutionMismatch
- setMatchesOnMultipleResolution:
- matchesOnMultipleResolution

Getting the representations
- bestRepresentationForDevice:
- representations
- removeRepresentation:

Determining how the image is stored
- setCachedSeparately:
- isCachedSeparately
- setDataRetained:
- isDataRetained
- setCacheDepthMatchesImageDepth:
- cacheDepthMatchesImageDepth

Determining how the image is drawn	– isValid – setScalesWhenResized: – scalesWhenResized – setBackgroundColor: – backgroundColor – setFlipped: – isFlipped – drawRepresentation:inRect: – recache
Assigning a delegate	– setDelegate: – delegate
Producing TIFF data for the image	– TIFFRepresentation – TIFFRepresentationUsingCompression:factor:
Managing NSImageRep subclasses	+ imageUnfilteredFileTypes + imageUnfilteredPasteboardTypes
Testing image data sources	+ canInitWithPasteboard: + imageFileTypes + imagePasteboardTypes

Class Methods

canInitWithPasteboard:

+ (BOOL)**canInitWithPasteboard:**(NSPasteboard *)*pasteboard*

Tests whether the receiver can create an instance of itself from the data represented by pasteboard. Returns YES if the receiver's list of registered NSImageReps includes a class that can handle the data represented by pasteboard.

NSImage uses the NSImageRep class method **imageUnfilteredPasteboardTypes** to find a class that can handle the data in *pasteboard*. When creating a subclass of NSImageRep that accepts image data from a non-default pasteboard type, override the **imageUnfilteredPasteboardTypes** method to notify NSImage of the pasteboard types your class supports.

See also: **+ imagePasteboardTypes**

imageFileTypes

+ (NSArray *)**imageFileTypes**

Returns an array of strings representing those file types for which a registered NSImageRep exists. This list includes all file types supported by registered subclasses of NSImageRep, plus those types that can be converted to supported file types through a user-installed filter service. The array returned by this method may be passed directly to the NSOpenPanel's **runModalForTypes:** method.

File types are identified by extension. By default, the list returned by this method contains "tiff", "tif", "bmp", and "eps".

When creating a subclass of NSImageRep that accepts image data from non-default file types, override NSImageRep's **imageUnfilteredFileTypes** method to notify NSImage of the file types your class supports.

See also: + **imageUnfilteredFileTypes**

imageNamed:

+ (id)**imageNamed:**(NSString *)*name*

Returns the NSImage instance associated with *name*. The returned object can be:

- One that's been assigned a name with the **setName:** method
- One of the named system bitmaps provided by the Application Kit

If there's no known NSImage with *name*, this method tries to create one by searching for image data in the application's executable file and in the main bundle (see NSBundle's class description for a description of how the bundle's contents are searched). If a file contains data for more than one image, a separate representation is created for each. If an image representation can't be found for *name*, no object is created and **nil** is returned.

The preferred way to name an image is to ask for a *name* without the extension, but to include the extension for a file name.

One particularly useful image is referenced by the string "NSApplicationIcon". If you supply this string to **imageNamed:**, the returned image will be the application's own icon. Icons for other applications can be obtained through the use of methods declared in the NSWorkspace class.

The image returned by this method should not be freed, unless it's certain that no other objects reference it.

See also: – **setName:**, – **name**, **– iconForFile:** (NSWorkspace), **+ imageFileTypes**

imagePasteboardTypes

+ (NSArray *)**imagePasteboardTypes**

Returns a null-terminated list of pasteboard types for which a registered NSImageRep exists. This list includes all pasteboard types supported by registered subclasses of NSImageRep, and those that can be converted to supported pasteboard types through a user-installed filter service.

By default, the list returned by this method contains “NSPostScriptPboardType” and “NSTIFFPboardType”.

When creating a subclass of NSImageRep that accepts image data from non-default pasteboard types, override NSImageRep’s **imageUnfilteredPasteboardTypes** method to notify NSImage of the pasteboard types your class supports.

See also: + **imageUnfilteredPasteboardTypes**

imageUnfilteredFileTypes

+ (NSArray *)**imageUnfilteredFileTypes**

Returns a null-terminated array of strings representing those file types for which a registered NSImageRep exists. This list consists of all file types supported by registered subclasses of NSImageRep, and doesn’t include those types that can be converted to supported file types through a user-installed filter service. The array returned by this method may be passed directly to the NSOpenPanel’s **runModalForTypes:** method.

See also: + **imageFileTypes**

imageUnfilteredPasteboardTypes

+ (NSArray *)**imageUnfilteredPasteboardTypes**

Returns a null-terminated list of pasteboard types for which a registered NSImageRep exists. This list consists of all pasteboard types supported by registered subclasses of NSImageRep, and doesn’t include those that can be converted to supported pasteboard types through a user-installed filter service.

See also: + **imagePasteboardTypes**

Instance Methods

addRepresentation:

– (void)**addRepresentation:**(NSImageRep *)*imageRep*

Adds *imageRep* to the receiver’s list of representations. After invoking this method, you may need to explicitly set features of the new representation, such as size, number of colors, and so on. This is true in

particular if the NSImage has multiple image representations to choose from. See NSImageRep and its subclasses for the methods you use to complete initialization.

Any representation that's added by this method is retained by the NSImage. Note that representations can't be shared among NSImages.

See also: – representations, **– removeRepresentation:**

addRepresentations:

– (void)**addRepresentations:**(NSArray *)*imageReps*

Adds each of the representations in *imageReps* to the receiver's list of representations. After invoking this method, you may need to explicitly set features of the new representations, such as size, number of colors, and so on. This is true in particular if the NSImage has multiple image representations to choose from. See NSImageRep and its subclasses for the methods you use to complete initialization.

Representations added by this method are retained by the NSImage. Note that representations can't be shared among NSImages.

See also: – representations, **– removeRepresentation:**

backgroundColor

– (NSColor *)**backgroundColor**

Returns the background color of the rectangle where the image is cached. If no background color has been specified, NSColor's **clearColor** is returned, indicating a transparent background.

The background color will be visible when the image is composited only if the image doesn't completely cover all the pixels within the area specified for its size.

bestRepresentationForDevice:

– (NSImageRep *)**bestRepresentationForDevice:**(NSDictionary *)*deviceDescription*

Returns the best representation for the device described by *deviceDescription*. If *deviceDescription* is **nil**, the current device is assumed. "Choosing Representations" in the class introduction outlines the process NSImage goes through in order to determine the "best" representation for a given device. For a list of dictionary keys and values appropriate to display and print devices, see NSGraphics.h.

See also: – representations, **– prefersColorMatch**

cacheDepthMatchesImageDepth

– (BOOL)**cacheDepthMatchesImageDepth**

Returns NO if the application's default depth limit applies to the off-screen windows where the NSImage's representations are cached. If window depths are instead determined by the specifications of the representations, **cacheDepthMatchesImageDepth** returns YES.

compositeToPoint:fromRect:operation:

– (void)**compositeToPoint:**(NSPoint)*aPoint* **fromRect:**(NSRect)*aRect*
operation:(NSCompositingOperation)*op*

Composites the portion of the image enclosed by the *aRect* rectangle to the location specified by *aPoint* in the current coordinate system. *aRect* must be a valid (non-null) rectangle. The *aPoint* argument is the same as for **compositeToPoint:operation:**. *op* should be one of the compositing operations as defined in **dpsOpenStep.h**.

The source rectangle is specified relative to a coordinate system that has its origin at the lower left corner of the image, but is otherwise the same as the base coordinate system.

This method doesn't check to be sure that the rectangle encloses only portions of the image. Therefore, it can conceivably composite areas that don't properly belong to the image, if the *aRect* rectangle happens to include them. If this turns out to be a problem, you can prevent it from happening by having the NSImage cache its representations in their own individual windows (with the **setCachedSeparately:** method). In this case, the window's clipping path will prevent anything but the image from being composited.

Compositing part of an image is as efficient as compositing the whole image, but printing just part of an image is not. When printing, it's necessary to draw the whole image and rely on a clipping path to be sure that only the desired portion appears.

See also: – dissolveToPoint:fromRect:fraction:

compositeToPoint:operation:

– (void)**compositeToPoint:**(NSPoint)*aPoint* **operation:**(NSCompositingOperation)*op*

Composites the image to the location specified by *aPoint* using the specified compositing operation, *op*.

aPoint is specified in the current coordinate system—the coordinate system of the currently focused NSView—and designates where the lower left corner of the image will appear. The image will have the orientation of the base coordinate system, regardless of the destination coordinates. *op* should be one of the compositing operations as defined in **dpsOpenStep.h**.

The image is composited from its off-screen window cache. Since the cache isn't created until the image representation is first used, this method may need to render the image before compositing.

When printing, the compositing methods do not composite, but attempt to render the same image on the page that compositing would render on the screen, choosing the best available representation for the printer. The *op* argument is ignored.

See also: – dissolveToPoint:fraction:

delegate

– (id)**delegate**

Returns the delegate of the NSImage object, or **nil** if no delegate has been set.

dissolveToPoint:fraction:

– (void)**dissolveToPoint:**(NSPoint)*aPoint* **fraction:**(float)*aFloat*

Composites the image to the location specified by *aPoint*, just as **compositeToPoint:operation:** does, but uses the **dissolve** operator rather than **composite**. *aFloat* is a fraction between 0.0 and 1.0 that specifies how much of the resulting composite will come from the NSImage. If the source image contains alpha, this operation may promote the destination NSWindow to contain alpha.

To slowly dissolve one image into another, this method (or **dissolveToPoint:fromRect:fraction:**) needs to be invoked repeatedly with an ever-increasing *aFloat*. Since *aFloat* refers to the fraction of the source image that's combined with the original destination (not the destination image after some of the source has been dissolved into it), the destination image should be replaced with the original destination before each invocation. This is best done in a buffered window before the results of the composite are flushed to the screen.

When printing, this method is identical to **compositeToPoint:operation:**. The *delta* argument is ignored.

dissolveToPoint:fromRect:fraction:

– (void)**dissolveToPoint:**(NSPoint)*aPoint* **fromRect:**(NSRect)*aRect* **fraction:**(float)*aFloat*

Composites the *aRect* portion of the image to the location specified by *aPoint*, just as **compositeToPoint:fromRect:operation:** does, but uses the **dissolve** operator rather than **composite**. *aFloat* is a fraction between 0.0 and 1.0 that specifies how much of the resulting composite will come from the NSImage. If the source image contains alpha, this operation may promote the destination NSWindow.

When printing, this method is identical to **compositeToPoint:fromRect:operation:**. The *aFloat* argument is ignored.

drawRepresentation:inRect:

– (BOOL)**drawRepresentation:**(NSImageRep *)*imageRep* **inRect:**(NSRect)*rect*

Fills the specified rectangle with the background color, then sends the *imageRep* a **drawInRect:** message to draw itself inside the rectangle (if the NSImage is scalable), or a **drawAtPoint:** message to draw itself at the location of the rectangle (if the NSImage is not scalable). The rectangle is located in the current window and is specified in the current coordinate system. This method returns the value returned by the **drawInRect:** or **drawAtPoint:** method, which indicates whether or not the representation was successfully drawn.

This method shouldn't be called directly; the NSImage uses it to cache and print its representations. By overriding it in a subclass, you can change how representations appear in the cache, and thus how they'll appear when composited. For example, your version of the method could scale or rotate the coordinate system, then send a message to **super** to perform this version.

If the background color is fully transparent and the image isn't being cached by the NSImage, the rectangle won't be filled before the representation draws.

initByReferencingFile:

– (id)**initByReferencingFile:**(NSString *)*filename*

Initializes the receiver, a newly allocated NSImage instance, for the file *filename*. This method initializes lazily: the NSImage doesn't actually open *filename* or create image representations from its data until an application attempts to composite or requests information about the NSImage.

filename may be a full or relative pathname, and should include an extension that identifies the data type in the file. The mechanism that actually creates the image representation for *filename* will look for an NSImageRep subclass that handles that data type from among those registered with NSImage. By default, the files handled are those with the extensions "tiff", "tif", "bmp", and "eps".

After finishing the initialization, this method returns **self**. However, if the new instance can't be initialized, it's freed and **nil** is returned. Since this method doesn't actually create image representations for the data, your application should do error checking before attempting to use the image; one way to do so is by invoking the **isValid** method to check whether the image can be drawn.

This method invokes **setDataRetained:**YES, thus enabling it to hold onto its file name. Note that if an image created with this method is archived, only the file name will be saved.

initWithBitmapHandle:

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

On Microsoft Windows platforms, **initWithBitmapHandle:** initializes the receiver, a newly allocated NSImage 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.

initWithContentsOfFile:

– (id)**initWithContentsOfFile:**(NSString *)*filename*

Initializes the receiver, a newly allocated NSImage instance, with the contents of the file *filename*. Unlike **initByReferencingFile:**, this method opens *filename* and creates one or more image representations from its data.

filename may be a full or relative pathname, and should include an extension that identifies the data type in the file. **initWithContentsOfFile:** will look for an NSImageRep subclass that handles that data type from among those registered with NSImage. By default, the files handled are those with the extensions “tiff”, “tif”, “bmp”, and “eps”.

After finishing the initialization, this method returns **self**. However, if at least one image representation can’t be created from the contents of the specified file, the receiver is freed and **nil** is returned.

initWithData:

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

Initializes the receiver, a newly allocated NSImage instance, with the contents of the data object *data*. If **initWithData:** is able to create one or more image representations, it returns **self**. Otherwise, the receiver is freed and **nil** is returned.

initWithIconHandle:

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

On Microsoft Windows platforms, **initWithIconHandle:** initializes the receiver, a newly allocated NSImage 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.

initWithPasteboard:

– (id)**initWithPasteboard:**(NSPasteboard *)*pasteboard*

Initializes and returns the receiver, a newly allocated NSImage instance, from *pasteboard*. *pasteboard* should contain a type returned by one of the registered NSImageRep’s **imageUnfilteredPasteboardTypes** methods; the default types supported are NSPostscriptPboardType (NSEPSImageRep) and

NSTIFFPboardType (NSBitmapImageRep). If *pasteboard* contains an NSFilenamesPboardType, the file name should have an extension returned by one of the registered NSImageRep's **imageUnfilteredFileTypes** methods; the default types supported are "tiff", "tif", "bmp", (all in NSBitmapImageRep) and "eps" (NSEPSImageRep).

If **initWithPasteboard:** is able to create one or more image representations, it returns **self**. Otherwise, the receiver is freed and **nil** is returned.

initWithSize:

– (id)**initWithSize:**(NSSize)*aSize*

Initializes the receiver, a newly allocated NSImage instance, to *aSize* and returns **self**. The size should be specified in units of the base coordinate system. Although you can initialize the receiver without specifying a size by passing a size of (0.0, 0.0) to **initWithSize:**, the receiver's size must be set before the NSImage can be used.

See also: – setSize:

isCachedSeparately

– (BOOL)**isCachedSeparately**

Returns YES if each representation of the image is cached separately in an off-screen window of its own, and NO if they can be cached in off-screen windows together with other images. A return of NO doesn't mean that the windows are, in fact, shared, just that they can be. The default is NO.

isDataRetained

– (BOOL)**isDataRetained**

Returns YES if the NSImage retains the data needed to render the image, and NO if it doesn't. The default is NO, except for images created with **initByReferencingFile:**, which should hold onto their file names. If the data is available in a file that won't be moved or deleted, or if responsibility for drawing the image is delegated to another object with a custom method, there's no reason for the NSImage to retain the data. However, if the NSImage reads image data from a file created with **initWithContentsOfFile:**, you may want to have it keep the data itself; for example, to render the same image on another device at a different resolution.

isFlipped

– (BOOL)**isFlipped**

Returns YES if a vertically flipped coordinate system is used when locating the image, and NO if it isn't. The default is NO.

isValid

– (BOOL)**isValid**

Returns YES if a representation for the receiver can drawn in the cache, and NO if it can't; for example, because the file from which it was initialized is non-existent, or the data in that file is invalid.

If no representations exist for the receiver, **isValid** first creates a cache with the default depth.

See also: – initByReferencingFile:

lockFocus

– (void)**lockFocus**

Prepares for drawing of the best representation of the NSImage for the current device by making the off-screen window where the representation will be cached the current window and a coordinate system specific to the area where the image will be drawn the current coordinate system. If the receiver has no representations, **lockFocus** first creates one with the default depth. See "Choosing Representations" in the class description for information on how the "best" representation is chosen.

A successful **lockFocus** message must be balanced by a subsequent **unlockFocus** message to the same NSImage. These messages bracket the code that draws the image.

If **lockFocus** is unable to focus on the representation, it raises an NSImageCacheException.

See also: **– bestRepresentationForDevice:**, **– isValid**, **– prefersColorMatch**, **– representations**

lockFocusOnRepresentation:

– (void)**lockFocusOnRepresentation:**(NSImageRep *)*imageRepresentation*

Prepares for drawing of the *imageRepresentation* representation by making the off-screen window where it will be cached the current window and a coordinate system specific to the area where the image will be drawn the current coordinate system. If *imageRepresentation* is **nil**, **lockFocusOnRepresentation:** acts like **lockFocus**, setting focus to the best representation for the NSImage. Otherwise, *imageRepresentation* must be one of the representations in the NSImage.

A successful **lockFocusOnRepresentation:** message must be balanced by a subsequent **unlockFocus** message to the same NSImage. These messages bracket the code that draws the image.

If **lockFocusOnRepresentation:** is unable to focus on the representation, it raises an NSImageCacheException.

See also: – isValid

matchesOnMultipleResolution

– (BOOL)**matchesOnMultipleResolution**

Returns YES if the resolution of the device and the resolution specified for the image are considered to match if one is an integer multiple of the other, and NO if device and image resolutions are considered to match only if they are exactly the same. The default is YES.

name

– (NSString *)**name**

Returns the name assigned to the receiver, or **nil** if no name has been assigned.

prefersColorMatch

– (BOOL)**prefersColorMatch**

Returns YES if, when selecting the representation it will use, the NSImage first looks for one that matches the color capability of the rendering device (choosing a gray-scale representation for a monochrome device and a color representation for a color device), then if necessary narrows the selection by looking for one that matches the resolution of the device. If the return is NO, the NSImage first looks for a representation that matches the resolution of the device, then tries to match the representation to the color capability of the device. The default is YES.

recache

– (void)**recache**

Invalidates the off-screen caches of all representations and frees them. The next time any representation is composited, it will first be asked to redraw itself in the cache. NSCachedImageReps aren’t destroyed by this method.

If an image is likely not to be used again, it’s a good idea to free its caches, since that will reduce that amount of memory consumed by your program and therefore improve performance.

removeRepresentation:

– (void)**removeRepresentation:**(NSImageRep *)*imageRep*

Removes and releases the *imageRep* representation from the NSImage's list of representations.

See also: – representations

representations

– (NSArray *)**representations**

Returns an array containing all of the representations of the receiver.

scalesWhenResized

– (BOOL)**scalesWhenResized**

Returns YES if image representations are scaled to fit the size specified for the NSImage. If representations are not scalable, this method returns NO. The default is NO.

Representations created from data that specifies a size (for example, the "ImageLength" and "ImageWidth" fields of a TIFF representation or the bounding box of an EPS representation) will have the size the data specifies, which may differ from the size of the NSImage.

See also: – setSize:

setBackgroundColor:

– (void)**setBackgroundColor:**(NSColor *)*aColor*

Sets the background color of the image. The default is NSColor's **clearColor**, indicating a transparent background. The background color will be visible only for representations that don't completely cover all the pixels within the image when drawing. This method doesn't cause the receiver to recache itself.

See also: – recache

setCacheDepthMatchesImageDepth:

– (void)**setCacheDepthMatchesImageDepth:**(BOOL)*flag*

Sets whether the application's default depth limit applies to the off-screen windows where the NSImage's representations are cached. If *flag* is NO (the default), window depths are instead determined by the specifications of the representations. This method doesn't cause the receiver to recache itself.

See also: **– lockFocus**, **– recache**

setCachedSeparately:

– (void)**setCachedSeparately:**(BOOL)*flag*

Sets whether each image representation will be cached in its own off-screen window or in a window shared with other images. If *flag* is YES, each representation is guaranteed to be in a separate window. If *flag* is NO (the default), a representation can be cached together with other images, though in practice it might not be.

If an NSImage is to be resized frequently, it's more efficient to cache its representations separately.

This method doesn't invalidate any existing caches.

See also: – recache

setDataRetained:

– (void)**setDataRetained:**(BOOL)*flag*

Sets whether the NSImage retains the data needed to render the image. The default is NO. If the data is available in a file that won't be moved or deleted, or if responsibility for drawing the image is delegated to another object with a custom method, there's no reason for the NSImage to retain the data. However, if the NSImage reads image data from a file that could change, you may want to have it keep the data itself. Generally, this is useful to redraw the image to a device of different resolution.

If an image representation is created using **initByReferencingFile:**, the only data retained is the name of the source file.

setDelegate:

– (void)**setDelegate:**(id)*anObject*

Makes *anObject* the delegate of the receiver.

setFlipped:

– (void)**setFlipped:**(BOOL)*flag*

Determines whether the polarity of the y-axis is inverted when drawing an image. If *flag* is YES, the image will have its coordinate origin in the upper left corner and the positive y-axis will extend downward. This method affects only the coordinate system used to draw the image; it doesn't affect the coordinate system for specifying portions of the image for methods like **compositeToPoint:fromRect:operation:** or **dissolveToPoint:fromRect:fraction:**.This method doesn't cause the receiver to recache itself.

See also: – recache

setMatchesOnMultipleResolution:

– (void)**setMatchesOnMultipleResolution:**(BOOL)*flag*

Sets whether image representations with resolutions that are integral multiples of the resolution of the device are considered to match the device. The default is YES.

setName:

– (BOOL)**setName:**(NSString *)*aString*

Registers the receiver under the name specified by *aString*, provided that no other NSImage is registered using that name. If the receiver is already registered under another name, **setName:** first unregisters the prior name. **setName:** returns YES unless another NSImage is registered using the name specified by *aString*, in which case **setName:** simply returns NO.

See also: + imageNamed:

setPrefersColorMatch:

– (void)**setPrefersColorMatch:**(BOOL)*flag*

Sets whether color matches are preferred over resolution matches when determining which representation to use. If *flag* is YES, the NSImage first tries to match the representation to the color capability of the rendering device (choosing a color representation for a color device and a gray-scale representation for a monochrome device), and then if necessary narrows the selection by trying to match the resolution of the representation to the resolution of the device. If *flag* is NO, the NsImage first tries to match the representation to the resolution of the device, and then tries to match it to the color capability of the device. The default is YES.

setScalesWhenResized:

– (void)**setScalesWhenResized:**(BOOL)*flag*

Sets whether representations with sizes that differ from the size of the NSImage will be scaled to fit. If flag is YES, representations are scaled to fit. The default is NO.

Generally, representations that are created through NSImage methods (such as **initByReferencingFile:**) have the same size as the NSImage. However, a representation that's added with either **addRepresentation:** or **addRepresentations:** may have a different size, and representations created from data that specifies a size (for example, the "ImageLength" and "ImageWidth" fields of a TIFF representation or the bounding box of an EPS representation) will have the size specified.

This method doesn't cause the receiving NSImage to recache itself when it is next composited.

See also: – setSize:

setSize:

– (void)**setSize:**(NSSize)*aSize*

Sets the width and height of the image. The size referred to by *aSize* should be in units of the base coordinate system.

The size of an NSImage must be set before it can be used. You can change the size of an NSImage after it has been used, but changing it invalidates all its caches and frees them. When the image is next composited, the selected representation will draw itself in an off-screen window to recreate the cache.

If the size of the image hasn't already been set when the NSImage is provided with a representation, the size will be set from the data. The bounding box is used to determine the size of an NSEPSImageRep. The TIFF fields "ImageLength" and "ImageWidth" are used to determine the size of an NSBitmapImageRep.

See also: **– initWithSize:**, **– setScalesWhenResized:**

setUsesEPSOnResolutionMismatch:

– (void)**setUsesEPSOnResolutionMismatch:**(BOOL)*flag*

Sets whether EPS representations are preferred when there are no representations that match the resolution of the device. The default is NO.

See also: – setMatchesOnMultipleResolution:

size

– (NSSize)**size**

Returns the size of the image. If no size has been set, and no size can be determined from any of the NSImage's representations, the returned NSSize will have a width and height of 0.0.

TIFFRepresentation

– (NSData *)**TIFFRepresentation**

Returns a data object containing TIFF for all representations, using their default compressions.

TIFFRepresentationUsingCompression:factor:

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

Returns a data object containing TIFF for all representations, using the specified compression type and compression factor. Legal values for *comp* can be found in **NSBitmapImageRep.h**, and are described in "Tiff Compression" in NSBitmapImageRep's class description. *aFloat* provides a hint for those compression types that implement variable compression ratios; currently only JPEG compression uses a compression factor.

If the specified compression isn't applicable, no compression is used. If a problem is encountered during generation of the TIFF, **TIFFRepresentationUsingCompression:factor:** raises an exception.

See also: **– TIFFRepresentationUsingCompression:factor:** (NSBitmapImageRep)

unlockFocus

– (void)**unlockFocus**

Balances a previous **lockFocus** or **lockFocusOnRepresentation:** message. All successful **lockFocus** and **lockFocusOnRepresentation:** messages (those that don't raise an NSImageCacheException) must be followed by a subsequent **unlockFocus** message. Those that raise should never be followed by **unlockFocus**.

usesEPSOnResolutionMismatch

– (BOOL)**usesEPSOnResolutionMismatch**

Returns whether EPS representations are preferred when there are no representations that match the resolution of the device. The default is NO.

See also: – matchesOnMultipleResolution:

Methods Implemented by the Delegate

imageDidNotDraw:inRect:

– (NSImage *)**imageDidNotDraw:**(id)*sender* **inRect:**(NSRect)*aRect*

Implemented by the delegate to respond to a message sent by the *sender* (an NSImage) when the *sender* was unable, for whatever reason, to composite or lock focus on its image. The delegate can:

- return another NSImage to draw in the *sender*'s place,
- draw the image itself and return **nil**, or

- simply return **nil** to indicate that *sender* should give up the attempt at drawing the image.