NXImage

Topics

Q: I'm using NXImage to display a PostScript or TIFF file. When I display it on a color system, the image doesn't look rightÐthere are large black areas.

Q: In my application I am reading in an NXImage. A nil is never returned, even if I read in a bogus file. Is this a bug? Here is my code:

Q: I'm writing an application which can open either EPS or TIFF images using the NXImage class. How can I determine what kind of file I've opened without hacking the file name?

Q: My application is a simple paint program. The user opens a TIFF image, then scribbles into it, and finally saves the new image as a TIFF file. However, the changes made by the user aren't saved into the TIFF fileÐit contains the original image. Why?

Q: I have allocated an instance of NXImage and an instance of NXBitmapImageRep. I then tell the NXImage to use the rep instance, like this:

Q: I'm using NXImage to display a PostScript or TIFF file. When I display it on a color system, the image doesn't look rightÐthere are large black areas.

A: Probably your image has transparency in it. The image was rendered into an NXImage and then composited onto the screen using **NX_COPY**. Since **NX_COPY** produces an exact copy of the bits from the source, transparent areas in the NXImage were copied onto the screen. On the monochrome MegaPixel display, these transparent areas expose to white, to emulate the way a sheet of paper might behave. NeXT's color devices act more like video devices, and they expose to black.

In order to avoid exposing the underlying device's representation of transparent, you should fill in the background and composite the NXImage using **NX_SOVER**:

```
- drawSelf:(NXRect *)rects :(int)rectCount
{
    NXPoint pt = {0.0, 0.0};

    NXSetColor(NX_COLORWHITE);
    NXRectFill(rects);
    [anImage composite: NX_SOVER toPoint: &pt];
   return self;
}
```

QA743

Valid for 2.0, 3.0

Q: In my application I am reading in an NXImage. A nil is never returned, even if I read in a bogus file. Is this a bug? Here is my code:

```
id myNXImage = [NXImage alloc];
if ([myNXImage initFromFile: "dummyName.tiff"] == nil)
{
       /* this is never getting called! */
       fprintf(stderr,"dummyName.tiff doesn't exist!\n");
}
```

A: This is not a bug. The **initFromFile:** method is lazy and does not catch *all* the errors that might happen when loading an image. Your application should be prepared to check for errors later on down the line either through delegation or by checking the **composite:** or **lockFocus** return values. If you wish, you can force the image to be rendered immediately:

```
id myNXImage = [[NXImage alloc] initFromFile: filename];
if ([myNXImage lockFocus])
       [image unlockFocus];
else
```

```
fprintf(stderr,"%s doesn't exist\n", filename);
```

Although this behavior might seem confusing it allows for more optimal performance: the image isn't rendered into the cache until it is needed. Rendering a large or complex file can be slowÐparticularly for a complex EPS file.

Note: Another good approach for determining whether an image can be successfully rendered is the NXImage delegate method **imageDidNotDraw:inRect:**. If you have assigned a delegate for the image and implemented this method, it gets called when compositing fails for whatever reason. See the documentation on NXImage for more information about this method. Also note that this method of delegation may be the only way to catch a drawing error for an image which is being "handed" to the AppKitÐan icon on a button, for example.

There is a known bug in Release 2 where **imageDidNotDraw:inRect:** fails to be called when encountering an error from within the method **composite:toPoint:**. This bug can be avoided by using the NXImage method **composite:fromRect:toPoint:**. This bug has been fixed in Release 3.

QA730

Valid for 2.0, 3.0

Q: I'm writing an application which can open either EPS or TIFF images using the NXImage class. How can I determine what kind of file I've opened without hacking the file name?

A: You can use the `isKindOf:` method from the Object class:

```
id myNXImage, myImageRep;

myNXImage = [[NXImage alloc] initFromFile: fileName];
myImageRep = [myNXImage lastRepresentation];
if ([myImageRep isKindOf: [NXBitmapImageRep class]])
{
       /* then I'm a TIFF file! */
}
else if ([myImageRep isKindOf: [NXEPSImageRep class]])
{
       /* then I'm an EPS file! */
}
```

The key here is that the NXImage instance itself does not understand EPS or TIFF information per se. NXImage manages the *representation* classes (one NXImage may have multiple representations) which do understand EPS and TIFF information.

Of course, it is reasonable to extract this information from the fileName as well. The following code snippet can be used to do this:

```
char *fileType = rindex(fileName, '.');

if (!fileType)
{
    /* then I'm not an appropriate file! */
}
else if (!strcmp(fileType, ".tiff"))
{
    /* then I'm a TIFF file! */;
}
else if (!strcmp(fileType, ".eps"))
{
    /* then I'm an EPS file! */
}
```

QA687

Valid for 2.0, 3.0

Q: My application is a simple paint program. The user opens a TIFF image, then scribbles into it, and finally saves the new image as a TIFF file. However, the changes made by the user aren't saved into the TIFF fileÐit contains the original image. Why?

A: This occurs if you open the TIFF file like this:

```
image = [[NXImage alloc] initFromFile:fileName];
```

NXImage will have two representationsÐthe file, and the cache. NXImage will treat the cache as a transitory image, and the file as its "best representation." The cache is the off-screen window to which the user's scribbles are drawn. When asked to write out the image, NXImage writes out its best representation of the imageÐwhich is the actual TIFF file residing on diskÐthus ignoring completely the changes made to the image. To get around this you must fake out NXImage by forcing the cache to be the best representation of the image.

The following code snippet illustrates what you must do:

```
/*  When the user opens the image */
rep = [[NXBitmapImageRep alloc] initFromFile:fileName];
[rep getSize:&imageSize];
```

```
image = [[NXImage alloc]   initSize:&imageSize];

if ([image useCacheWithDepth:d] && [image lockFocus]) {
        [rep draw];
        [image unlockFocus];
}
[rep free];
```

This code sample initialized an NXBitmapImageRep from the file containing the opened image. The NXImage is initialized from this representation. Now the NXImage does not have a file which can serve as its best representationÐit only has the cache. Thus when you tell NXImage to **writeTIFF:** the cache with all of the user's scribbles is written out properly.

QA786

Valid for 1.0, 2.0, 3.0

Q: I have allocated an instance of NXImage and an instance of NXBitmapImageRep. I then tell the NXImage to use the rep instance, like this:

```
NXRect originalSize;
id myRep, myImage;
```

```
int bitsPerPixel;

myRep = [[NXBitmapImageRep alloc] initFromFile: filename];
[myRep getSize: &originalSize];
myImage = [[NXImage alloc] initSize: &originalSize];
[myImage useRepresentation: myRep];
```

Then, later in my application I query the rep instance (as follows) and the query fails because myRep is nil! Why is this?

```
bitsPerPixel = [myRep bitsPerPixel]; /* this fails -- myRep is nil ! */
```

A: This is not a bug. Once you have ªgivenº the NXBitmapImageRep instance to NXImage (by calling `useRepresentation:`) then the NXImage "owns" that rep and can do what it wishes with it. (This is also true for any class of rep instance, not just NXBitmapImageRep) What the NXimage typically does is to turn that representation into an NXCachedImageRep and then free the NXBitmapImageRep. To prevent this behavior do a `setDataRetained:YES` on the NXImage instance. The `setDataRetained:` method defaults to NO. The NXImage then does not free the NXBitmapImageRep. For example, to correct the above example, add the following line prior to calling `useRepresentation:`

```
[myImage setDataRetained:YES];
```

QA732

Valid for 2.0, 3.0