TIFFandEPS

by Sharon Zakhour, NeXT Developer Support Team

Overview

This example shows how to open EPS or TIFF images and save them back out as EPS or TIFF -- converting the image in the process. The example was originally written to convert from EPS-format images to TIFF-format images. The implementation used draws the image into an instance of ImageView. This view is then inserted as the contentView of the window created by ImageReader to the exact size of the image. The TIFF version is generated using the `initData:fromRect:` method of NXBitmapImageRect. This method allows you to create bitmap data from anywhere on the screen quickly and easily but has some inherent drawbacks: as the bits are being read from the windowserver backing store the resulting image is limited to the window resolution and will contain any dithering artifacts generated by the windowserver.

Excerpted from the `initData:fromRect:` description in the NXBitmapImageRep spec sheet:

> This method uses PostScript imaging operators to read the image data into the *data* buffer; the object is then created from that data. The object is initialized with information about the image obtained from the Window Server.

The NXImage classes do not contain support for converting an image to EPS so the View method `copyPScodeInside:to:` is used to convert in this direction.

Disclaimer

This app is best suited to converting EPS to TIFF and TIFF to EPS. When saving an EPS file which was read in as an EPS file [or likewise when saving a TIFF read in as a TIFF] no intelligent decisions are made in the code. That, of course, is left to the reader. The result [in both cases] would be to create an EPS or TIFF file that contains any dithering employed by the window server and would not be identical to the original.

Program Organization

How to build the nib files

There is one nib file in this example:

TIFFandEPS.nib	The main nib file contains the main menu and the classes. The file's owner for this nib file is ImageReader -- a subclass of Object and the application delegate. The nib file also contains an instance of ImageReader and the window containing the accessory view for the SavePanel.

Classes in the Application

ImageReader	Subclass of Object. This class serves both as the application delegate and the delegate of all windows that it creates. The `openRequest:` method is called when the user selects "Open Image..." from the menu. Initially the "Save Image..." menu item is disabled. When a window is opened with an EPS or TIFF image, the "Save Image..." menu item becomes enabled. The object is able to detect this because it has implemented the window delegate methods `windowWillClose:` and `windowDidBecomeMain:`. The `saveRequest:` method is called when the user selects "Save Image..." from the menu. Each window is created to the exact size of the image -- ScrollViews are not used.
ImageView	Subclass of View. This class is pretty simple. The `initFromImage:` method initializes an instance of this class and saves the NXImage instance into an internal instance variable. The `drawSelf::` method composites the entire NXImage instance. One of the advantages of using NXImage over the representation classes (like NXBitmapImageRep) is that it maintains an offscreen cache that supports compositing -- much faster than drawing the representation each time.

PopAndForm	A little object that uses a Popup and a FormCell together. This object is hooked up to a FormCell and a PopUp so that the form can be used to add options to the Popup. As far as actions go, the "Specify" item in the popup menu needs to be hooked to "enableForm:" in the object and the Form needs to be hooked to "newValue:" as well.

Topics Of Interest

How to implement an accessory panel
The accessory panel feature allows a quick and easy mechanism for customizing any of the standard panels. In this example the accessory panel allows the user to select what format to use when saving the image: EPS or TIFF. And if a TIFF image the user may select the resolution and the compression type. Because this view contained quite a number of controls I visually separated it from the rest of the save panel using a box that I placed underneath all of the controls. I then increased its width to be wider than the view (cropping the vertical lines). This gives the pleasing effect of two horizontal lines above and below the view. The accessory panel is installed in the save panel with the `setAccessoryView:` method.

How to enable/disable menu items
Initially the "Save Image..." menu item is disabled in IB. This menu item becomes enabled at run time when any EPS or TIFF images are opened. This is done using the window delegate methods `windowWillClose:` and `windowDidBecomeMain:`. The `windowWillClose:` method is called when a window is *about* to close. Examining the current window count determines whether the user is about to close the last remaining window -- and if so, the "Save Image..." menu item is again disabled. Likewise `windowDidBecomeMain:` is invoked when any window becomes the main window and the "Save Image..." menu item is then enabled. This does cause the menu item to be redundantly enabled at times but is harmless and has no visual side effects.

How to enable/disable items in the accessory view of the Save Panel
The accessory view for the save panel contains three radio button matrices. The TIFF vs. EPS matrix causes all other controls in the view to be invalid when the EPS item is selected. This is backed up

with the visual cue of enabling/disabling the other controls as appropriate. The `selectFormat:` method in ImageReader is the target action of the TIFF/EPS matrix. It determines whether it should enable or disable the other controls in the view. It also sets the required file type on the SavePanel to the appropriate extension -- .eps or .tiff.

Within the TIFF world, the user may select from the DPI matrix [one of the selections allows the user to specify a nonstandard DPI] and from the compression matrix -- LZW vs. JPEG. When LZW is selected the JPEG compression factor field is invalid. So a second level of control enabling/disabling is implemented to handle this. The `selectCompression:` method in ImageReader is the target action of the LZW/JPEG matrix. In fact, when the user selects TIFF in the first matrix the `selectFormat:` calls the `selectCompression:` method to determine what state it should re-set the JPEG controls to.

How to use an alert panel
Most of the error conditions in this example are handled using the perror(3) utility. However the alert panel feature is used when the user attempts to save a TIFF file using JPEG compression on an image unsuitable for JPEG. The `NXRunAlertPanel()` function throws the app into a modal loop.

Other Files

TIFFandEPS_main.m, Created by Interface Builder.
IB.proj,
Makefile,
TIFFandEPS.iconheader

Bugs

This app can tickle a bug in 2.0 when converting from EPS to TIFF:
Bug #13606 has been described as an ugly line at the right end of images when printing and occurs if a TIFF file is 2-bit grayscale, has alpha, is not a multiple of 4 pixels wide, and has a "bad" alpha at the end of the scanline. User workarounds: Make sure your TIFF doesn't have alpha, or is a multiple of 4 pixels wide, or doesn't have illegal alpha. The last item is hard to do (once the image contains the bad alpha), but the others aren't so bad...