Programmer's Stuff

So you'd like what this object does and why you'd want to use it? Well, this will tell you how, but first a few brief words about the source. The main documention is in the header files at the moment (where about the only comments appear). They will tell you about what a method expects, however, this will tell you how they relate to each other. The final release will have much more comprehensive documentation.

Overview

You might also want to know what this program does, I suppose. Well, it's sort of a wrapper that fits around an NXBitmapImageRep, thus the object name NXBitmapImageRepControl. It's purpose is to make the NXBitmapImageRep a more useful object by supporting more types of graphics formats. It does this through calls to *objc_loadModules()* and

objc_unloadModules() (NeXStep Refernce, Volume 2, page 3-157). So, when you request it to load a module, it looks out on disk for a converter (called thus since it converts the format to the internal NeXT format and back again). It reverses the process when saving and image.

Part of the goal of these object was to allow the most flexibility in upgrading the objects with the least pain to the programmer. To this end, the only object actally linked into the program at compile time is the ConvertLoader object. Messaging/instantiating this object loads a Control object off the disk. This is currently only availble for bitmaps, but may yet become availble for other file formats, like sounds.

Once the Control object is linked, it links in the ImageControl object, which is used by converters to convert between various bitmap formats, and to get pixel information out of a bitmap. It also scans the library folders and makes a list of all known converters and their file types.

Converters
Now, what's in a converter? A converter is basicially an unlinked object module, created with a *cc -c <filename> -o <format.extension>.bcvt* system call, and possibly a *ld -r <source file>¼*, which ªpre-linksº a set of .o files and libraries. The only realy requirement is that the converter must always be first in the final mach-o file.

The object is of type Converter and all object of this type must respond to certain messages in a predescribed way. Meeting these specifications fully will allow programs to easily access many types of bitmap formats. You can find a template for writing a converter object in ¼/GraphicsWorkship/Converters/template.[hm].

The converters are stored in library folders about the file system. They will be using in the following order: ~/Library/Converters is first, followed by a converter in /LocalLibrary/Converters, which is followed by /NextLibrary/Converters. All recognized converters should have the *.bcvt*

suffix. The name of the converter is also important as it's used to get the file type. Therefore, a TIFF converter would be called *tiff.bcvt*.

Image Manipulation Tools

Also, since many of these converters need a common set of routines and utilities for manipulating images, they may request an object to do this for them by message their sender. This is usually the NXBitmapImageRepControl object. This object is also stored on disk in a */library/*Converter folder under the name of *Image.tools*. The object contains a list a standard functions for converting bitmaps from color to black and white to getting individual pixels from an image. Because this program is linked at run time, it may be modified at a later date to improve the functionality of the routines and it's algorithms. This can be especially important, since this is the prime place something might cause a crash or need to be change to increase functionality.

Of course, these tools are availble to any application programmer who may

also decide to rewrite them.

The Steps to Using It

Now that you know what it uses, here's how to use it. For examples of any of these steps, see the source to *GraphicsWorkshop*.

1. Initialize the ControlLoader object by telling it to load an instance of "Bitmap" tools. This will cause the program to search for converters, as well as link in the ImageControl object for later instantiations. It should only be called once.
2. When ready, create an open panel, but to run it, call the *runOpenPanel* method in the NXBitmapImageRepControl object. This will show the open panel for all file types found in Step 1. It will return the same values as the normal Open Panel found in the NeXTStep Reference, Volume 1.
3. Once you have from 1 to *n* filenames, pass these names, one at a time, to the *openAndReturnImage* method. This will return an NXBitmapImageRep for each image.

4. Manipulate the image any way you'd like.
5. Now that you're ready to save the image, create a save panel, but run it via the *runSavePanel* method. This will bring up a panel, with the images current type linked into the code. Should the user select a new type, that converter will be linked, it's custom view, if any, displayed, and it's file extension added to the bitmaps filename. If it returns YES, then you can move onto step 6, otherwise, the user canceled the save.

5a. The programmer can also get parameter information from the converter object itself at this point. However, this is discouraged in applications using the NeXTStep interface, as every converter can have it's ªownº set of parameters and using the set and get methods only ties the program down to a few or one specific converter.

6. Call the *saveImage* method to save the bitmap. The correct converter was already linked, so make sure you don't call the *handleLink* message before the *saveImage* message as this will eradicate the user's customazation selections.
7. That's it.

Some Pointers

- You can call the routines to link converter maually. This is useful when needing to bi-pass the open and save panels or working at the UNIX shell level.
- Most converters should work without the WindowServer active. Examples of ones that might not is one that reads EPS files.
- File extensions are *very* important. This is how the object can tell what kind of converter is being used. Make sure, when passing names, that the filename always has a valid extension, otherwise thing won't work nicely for the user.
- When running a save panel, the linked converter's specializeation parameters only last as long as the next save panel.
- If you'd like to open multiple images in a file, you'll have to link converters yourself and message them via the openMulitple and saveMultiple calls.

Pointers on Writing Converters

- You need not implement any of the routinse, however, you must respond to all messages up to the version level you're programming.
- It's really kind of pointless to not implement Reading and Writing of single images to streams, however, should you wish to not do this, simply make sure to always return NO for the one not implemented.
- If you don't wish to implement the "all" routines, also return NO, however, it is preferable that this return a single image inside and NXImage object, if the format only supports one image per file. If it supports multiple images, this should definitely be implemented (of course, it's not for gif :-(, or any other routines for that matter).
- You only need to implement customSaveView and customOpenView if you'd like, but always return nil when you don't. The width parameter is the max width of the save panel. This helps when laying things out.
- Init and free are always called, however, they don't need to do anything unless you need to.
- The version method really needs to return something valid. It's assumed all converters will always respond to protocol level 1.0. However, should

new messages be implemented for ªunseenº bitmap features in the future, this protocol level may increase, which is why it's important that you return the string ª1.0º, as it's the only way that NXBitmapImageRepControl knows what not send your converter in the future.

- The ability to handle errors has been made more robust. Please see sample code and header files for more details.