NXBitmapImageRepControl

INHERITS FROM	Object
DECLARED IN	NXBitmapImageRepControl.h

CLASS DESCRIPTION

This object is the object you should be talking to the most. It's job it to be the go between for you program and the converters stored on disk. Like the converters, however, this object is also stored on disk, so feel free to modify the code and place a new copy in the Converters directory as Bitmap.controls. I do ask a couple of things. First, if you think a change warrants it, such as a major bug fix, please send me the modified code, so that I can consider it for release in a future release or a patch. Secondly, I ask that you do not distribute modified version. This has the possibility of creating a lot of confusion and difficulty for the user.

Using this object is basically fairly simple. You simply need to instantiate a new version via the use of the ControlLoader object. From there, you ask it to get a new filename via it's open panel. It can then use this filename to open a new bitmap. Doing the reverse by calling the save panel, and then passing the filename bask to the program can save an image.

INSTANCE VARIABLES

Inherited from Object	Class	isa;
Declared in Converter	int	errorState
	int	typesCount
	int	curMax
	Type	**types
	Type	*curType
	BOOL	useNeXTStep
	id	myView
	BOOL	amSaving
	id	currentPanel
	id	currentView
	id	myBox
	NXRect	viewDefault
	id	mySubView
	id	panelSave
	id	myPopUp
	id	myButton
	id	nameText
	char	picName[MAXPATHLEN]

typesCount	Hold the number of converters found in the various library folders.
curMax	This holds the current maximun number of types. This value will grow, dynamically, if many converters are found.
**types	A dynamically growing array of types found on disk.

*curType	Holds the current linked type, or **NULL** if nothing is linked.
useNeXTStep	If **YES,** then the object will attempt to use the window server where appropiate. If this is set incorrectly, it can cause premature program termination.
myView	Used to hold my custom view used in the open and save panels.
amSaving	If **YES,** then the current operation is saving, otherwise it is opening. This is used by both run methods for open and save panels.
currentPanel	Holds the **id** of the currently running panel.
currentView	Holds any custom view handed in by the application programmer along with their save or open panel.
myBox	One of the controls used in myView.
viewDefault	Default size of myView.
mySubView	Used when a converter has a subview to hold that **id**.
panelSave	Used to pass around the **id** of the save or open panel.
myPopUp	One of the controls used in myView.
myButton	One of the controls used in myView.
nameText	One of the controls used in myView.
picName	Current name of the bitmap. This is really only valid after the *runSavePanel:withName:* method has been called.

METHOD TYPES

Creating and Freeing a Converter	- init - free
Reading and Writing	- runOpenPanel: - openAndReturnImage: - runSavePanel:withFilename: - saveImage:toFile:
Error Handling	- errorState - errorMessage - errorStringMessage
Linking and Unlinking	- handleLink: - unlinkConverter
Informational Querries	- getTypeList - getCurrentFormatName - getCurrentConverter ± getImageControl: ± filename ± usesNeXTStep ± setUseNeXTStep:

INSTANCE METHODS

errorState

- (int)**errorState**

Returns the current error state of the converter. The type of error state return can be found in Converter.h, and must be CONVERT_ERR_NONE, CONVERT_ERR_WARNING, or CONVERT_ERR_FATAL. The image should be returned unless the error state reported is CONVERT_ERR_FATAL.

See also: - **errorMessage**
- **errorStringMessage**

errorMessage

- (int)**errorMessage**

Return a standard error code. These are defined in Converter.h and remain constant. If the converter needs to return a non-standard error message, you will need to call *errorStringMessage* to get the exact error code, however this is the prefered method, since it allows for internationalization.

See also: - **errorState**
- **errorStringMessage**

errorStringMessage

- (char *)**errorStringMessage**

Returns a **NULL** terminated string describing the current error state of the converter. This string should be usable in a dialog box which informs the user of the current problem.

See also: - **errorState**
- **errorMessage**

filename

- (char *)**filename**

This returns a **NULL** terminated string describing the current filename being used by the control object. Normally only the converters will call this message. It's here so that can find out the name. As far as the calling program is concerned, it gave the control object this name in the first place via a message to the *runSavePane:withName:* message.

free

- **free**

Frees storage use by the object. After calling this method, you can no longer message the object reliably.

getCurrentConverter

- **getCurrentConverter**

Returns the *id* of the current converter. This can then be used to speak with a converter directly. You can only call the method if you have previously linked a converter. You can accomplish converter linking by calling *runSavePanel:withName:*, *runOpenPanel:*, or *handleLink:*. Otherwise the return value has no meaning.

getCurrentFormatName

- (char *)**getCurrentFormatName**

Assuming that a converter has been linked, this method return the name describing the current format. For example, if the TIFF converter were linked, this would return ªTIFF (Tagged Image File Format)º. It returns a **NULL** terminated string. This string is only valid for the life of the linked converter, so you should make a copy of it if you'd like to keep it around.

getImageControl:

- **getImageControl**: (id)*image*

This is here primarily for the converters to call. They use this method to get an instance of the **ImageControl** object. However, your program may also choose to use this method if you need a copy of that object. *Image* is an **NXBitmapImageRep** and will be associated with the return **ImageControl** object.

getTypeList

- (char **)**getTypeList**

This returns an array of character pointers which point to **NULL** terminated character strings which represent the current, usable image types. The last item in the list will be **nil**. You may pass any of these strings to the *handleLink* method. The list will be valid until the **NXBitmapImageRepControl** object is freed.

When done with this list, you should make sure to deallocate it's storage using **free (3).**

handleLink:

- **handleLink:** (char *)*inType*

Attempts to link the converter for *inType*. If it fails to link that converter, this method return **nil**, otherwise it returns an id to an instance of that converter. You may only have one converter linked at a time, so it you received a previous instance of a Converter, it will be unlinked, and the id you already have will be invalid. This should perhaps not work this way, but it will until I can figure out how to allow multiple converters to be linked simultaneously.

init

- **init**

Initializes the control object.   This method will attempt to get the base object **ImageControl**, and to also find all converters located in the library directories. If it's successful, it will return self, otherwise it returns **nil**.

openAndReturnImage:

- **openAndReturnImage:** (const char *)*filename*

Returns an **id** to a new instance of a **NXBitmapImageRep** that contains the bitmap data for the image type found in filename. *filename* is a **NULL** terminated string that represents the file you wish to load. It should contain the document extension for it's file type. For example, a give image would be in the form *foo.gif.* If the file extension is not present, results can be unpredictable. If the image could not be loaded, **nil** will be returned, and the caller should then message the *errorState* method to find out the problem. You should probably send this message, anyways, to check for warning conditions.

runOpenPanel:

- **runOpenPanel:** (id)*openPanel*

Runs the open panel asking the user to input a list of filenames. You will always receive back a list of filenames from the user, however, you may choose to only use the first filename found. The user will be presented with a close to standard open panel containing a custom view to allow them to select specific file types or any file type they wish. You can get back a mixed list of file types.

OpenPanel is an instance of your applications open panel and should be gotten via a call to the **OpenPanel**'s **new** method. You may insert a custom view into the open panel before passing it along. This panel will be incorporated in the control object's custom view, allowing your application to request addional information from the user.

This object returns the same values as the open panel. **self** if successful or **nil** if not.

runSavePanel:

- **runSavePanel:** (id)*savePanel*

This is similar to the *runOpenPanel* method, but runs a standard save panel. Like the normal save panel, it only allows the user to select on filename at a time. Also like the *runOpenPanel* method, this will allow the use of your own custom views.

This method accepts the application's save panel via *savePanel* and will return it's values on exit.

saveImage:toFile:

- (BOOL)**saveImage**: (id)*image*
  toFile: (const char *)*filename*

Saves the image *image* to the file *filename*. If you need to save an image to something else, you will need to message the converter directly. *filename* should contain a valid document extension, so that the converter knows what file format the user wants the image save under. Valid filetype extensions can be found via the *getTypeList* method. This method return **YES** if the image was successfully written to disk, other wise it returns **NO**.

setUseNeXTStep:

- **setUseNeXTStep**: (BOOL)*state*

This sets whether or not the control object should attempt to use the window server. This is relevant for things like how to inform the user of problems. If set to **YES** it will run alert panels to inform the user of problems, otherwise it will print the errors to **stderr**.

Returns **self**.

unlinkConverter

- **unlinkConverter**

Explicity unlinks whatever converter is currently linked. Normally you will not need to call the method, however, if you program is doing dynamic run time linking of other objects, you should always make sure that this method is called before anything else is linked. This ensures that your other objects will not be unlinked by mistake. This is a short comming of the run time linker system that only allows the previously linked object to be unlinked.

Returns **self**.

usesNeXTStep

- (BOOL)**usesNeXTStep**

Returns **YES** if it thinks it can use the window server, otherwise it return **NO**.

CONSTANTS AND DEFINED TYPES

```
typedef struct {
   char   fullpath[MAXPATHLEN];
   char   type[MAXPATHLEN];
} Type;
```