DynamicApplication

INHERITS FROM	Application : Responder : Object
DECLARED IN	DynamicApplication.h

CLASS DESCRIPTION

The DynamicApplication class provides added functionallity to applications that would like to dynamically load Classes (eg. Palettes and Interface Builder). Applications that inherit from DynamicApplication will be able to dynamically load Classes and use those Classes in their running program.

The ability to dynamically load Classes has many advantages. The programmer can write his/her application wide open, so that the program is not limited to solely his/her thinking. For instance Interface Builder supplies the user with a wide viriety of objects to deal with, but it also allows the user to create his/her objects and link them into the running version of Interface Builder (Palettes). The creator of a graphics package could leave his/her program open to new file types that haven't even been thought of yet. He/She could supply objects that know how to interpret tiff, pict, eps, etc.., but as the graphic formats change, the programmer would only need to write objects that know how to interpret an new file type, and then the user could just load one of these classes into the existing version of the program. The original programmer would not have to constantly keep up with every graphic type in the world. He can right a robust

program, and leave tweeking to the users. To speculate on exactally what could be achived with this Class is beyond the scope of this programmer 8-).

To use this object is fairly straightforward, but there are a few things that have to be done in order to load Classes. First, the classes that you are going to load have to be compiled. This version of DynamicApplication only loads one Class at a time so object (".o") files should only contain one class. I have found that the following produces object files that are compatible with DynamicApplication.

```
cc -g -c -Wall [SomeClassName].m -o [SomeClassName].o
```

Second, instances of the classes that are going to be loaded should implement the **-name** method. This method is used to name the Classes. If the object does not implement the **-name** method, then DynamicApplication tries to name it itself, and it can come up with rather bizarre names.

The last and most important thing to remember when loading Classes is that you are dealing with Classes, not Objects. So in order to create instances you most send **+new** or **+alloc** messages to the Class that you just loaded.

INSTANCE VARIABLES

Inherited from Object	Class	isa;
Inherited from Responder	id	nextResponder;
Inherited from Application	char	*appName;
	NXEvent	currentEvent;
	id windowList;	

```
id    keyWindow;
id    mainWindow;
id    delegate;
int   *hiddenList;
int   hiddenCount;
const char                  *hostName;
DPSContext                  context;
int   contextNum;
id    appListener;
id    appSpeaker;
port_t                      replyPort;
NXSize                      screenSize;
short                       running;
struct __appFlags           appFlags;
```

Declared in DynamicApplication

```
id                          dynamicClasses;
int                         dynamicClassesNum;
struct _errorFlags {
    BOOL                        _errorOnLastLoad;
    BOOL                        _classLoaded;
    unsinged int                _errorType;
}                           errorFlags;
struct _lastClassInfo {
    char                        *_className;
}                           lastClassInfo;
```

dynamicClasses A HashTable containing the names of classes as keys and the pointers to

	associated Class structures as values. This HashTable is initially empty and is filled as new Classes are loaded into the running application.
dynamicClassesNum	The number of Classes that have been loaded into the running application.
errorFlags._errorOnLastLoad	Set to YES, if there were problems loading the last Class.
errorFlags._classLoaded	Set to YES, if a Class was succesfully loaded by the most recent call to **-loadClass:** or **-loadClass: withName:**.
errorFlags._errorType	The type of the most recent error.
lastClassInfo._className	The name of the Class that has most recently been loaded.

METHOD TYPES

Initializing the class	+ initialize
Creating and freeing instances	+ new - free
Loaded new Classes	- loadClass: - loadClass:withName:
Accessing new Classes	- classWithName: - getClassNameList: - classesNum
Handling Errors	- errorOnLastLoad:

- errorType:
- classLoaded:

Removing Class references - removeClassWithName:

CLASS METHODS

initialize

+ **initialize**

Registers defaults used by the DynamicApplication class. You never send this message directly; it's sent for you when your application starts. Returns **self**.

new

+ **new**

Creates a new DynamicApplication object and assigns it to the global variable **NXApp**. A program can have only one DynamicApplication or Appliation object, so this method just returns **NXApp** if the an Application object already exists. This method also makes a connection to the Window Server, loads the PostScript procedures the DynamicApplication needs, and completes other initialization. Your program should generally invoke this method as one of the first statements in **main()**; this is done for you if you create your application correctly with Interface Builder. If you are using Interface Builder to create your Application, you will need to take the following steps to ensure that DynamicApplication is being used correctly: 1) Include the DynamicApplication.[hm] files in your project. 2) Subclass Application, and name that subclass "DynamicApplication" 3) Change the File's Owner from Application to DynamicApplication (in Attributes). The DynamicApplication object is returned.

INSTANCE METHODS

classesNum

- (int)**classesNum**

Returns the number of classes that the Application has dynamically loaded since startup.

See also: - **getClassNameList:**

classLoaded

- (BOOL)**classLoaded**

Returns YES, if the last call to **-loadClass:** or **-loadClass:withName:** succesfully loaded a Class description, otherwise returns NO.

See also: - **loadClass, - loadClass:withName:**

classWithName:

- **classWithName:**(char *)*className*

Returns the Class structure identified by the name *className*. For this to return the expected value, the Class associated with *className* would have to be a dynamically loaded Class that has already been loaded with either the **-loadClass:** or **-loadClass:withName:** methods. This method will not return structure of Classes that have not been dynamically loaded into the running application. For instance your application might know of the "Box" class when it was compiled, but this method will not return a pointer to the Box Class, unless you dynamically load a different Class structure that you call "Box".

See also: - **loadClass, - loadClass:withName:**

errorOnLastLoad

- (BOOL)**errorOnLastLoad**

Returns YES, if the last call to **-loadClass:** or **-loadClass:withName:** produced an error message, otherwise returns NO.

See also: - **loadClass,** - **loadClass:withName:**

errorType

- (unsigned int)**errorType**

Returns an integer representing what type of error occured during the last call to **-loadClass:** or **-loadClass:withName:.** For a list of all known errors, see the end of this document.

See also: - **errorOnLastLoad**

free

- **init**

Closes all the DynamicApplication object's windows, breaks the connection to the Window Server, and frees the DynamicApplication object.

getClassNameList:

- **getClassNameList:**(char **)*classList*

This methods places an array of strings in the memory pointed to by *classList*. It is assumed that sufficent memory has been allocate to *classList* before being passed into this method. The array of strings placed into *classList* represent the names of all the Classes that have been loaded into the current application.

See also: - **classesNum**

init

- **init**

Clears the instance variables of the DynamicApplication object. Sets the number of dynamically loaded Classes to 0, and creates a new HashTable to hold the classes and their names.

lastClass

- **lastClass**

Return the Class description of the last Class that was successfully loaded by a call to **-loadClass:** or **-loadClass:withName:**.

See also: - **lastClassName**

lastClassName

- (char *)**lastClassName**

Return the name of the last Class that was successfully loaded by a call to **-loadClass:** or **-loadClass:withName:**.

See also: - **lastClass**

loadClass:

- **loadClass:**(char *)*classPath*

Loads a class description from the object file *classPath*. The string *classPath* has to be the filename of a ".o" file that

contains the object code describing a Class structure. If a Class instance responds to **-name** then the Class is placed into the Class HashTable based on the name of its instance.

See also: - **loadClass:withName:**

loadClass:withName:

- **loadClass:**(char *)*classPath* **withName:**(char *)*className*

To be used if you don't know the name of the Class that you are going to load. If the Class responds to **-name** then the application will first assign it's true name to the loaded Class. It will then make a second reference to the loaded Class. So if you use this method to load a new Class, then you will be able to access the class by two names, both the Classes "true" name, and *className*.

See also: - **loadClass:**

removeClassWithName:

- **removeClassWithName:**(char *)*className*

Remove the Class with the name *className* from the application. In this version of the class, *className* is just removed from the HashTable, it is not actually removed from the application. So any object that have been created by Classes that you want to remove, will still function correctly after the Class has been removed.