Executive

INHERITS FROM	Object
REQUIRES HEADER FILES	Executive.h
DEFINED IN	

CLASS DESCRIPTION

Executive allows the execution of shell commands from a program, providing both synchronous and asynchronous execution of commands. Other features include the ability to direct command output to another object through the popen(3) mechanism and displaying of errors through a standard panel for consistency in error reporting throughout an application.

INSTANCE VARIABLES

Inherited from Object	Class	isa;
Declared in Executive	id	target;
	SEL	action;
	double	period;
	int	curCmdId;
	int	numExecuting;
	mutex_t	runningLock;
	mutex_t	doneLock;
	id	running;
	id	done;

target	The target of asynchronous operations
action	Action to be sent to the target when an asynchronous operation ends
period	Period at which the timed entry gets called
curCmdId	Used internally for tracking command identifiers
numExecuting	Used internally to track the number of asynchronous commands executing

runningLock	Mutual exclusion for the running queue
doneLock	Mutual exclusion for the done queue
running	Queue of commands running asynchronously
done	Queue of asynchronous commands that have finished

METHOD TYPES

Creating and freeing instances	- free + new + newPeriod:
Target and Action	- setTarget: - target
Setting timed entry period	- setPeriod: - period
Executing commands	- execute: - execute:async: - execute:async:environs:
Reading pipes	- pipe:to:: - pipe:environs:to::
Showing errors	- showError: - showError:while: - showError:while:on: - showError:while:on:using:

CLASS METHODS

new

+ new

Creates a new Executive object with the default period for updating the timed entry.

See also: **newPeriod:**

newPeriod

+ newPeriod:(double)*period*

Creates a new Executive object with *period* for updating the timed entry.

INSTANCE METHODS

target

– target

Returns the target of the Executive

See also: **setTarget:**

setTarget:

– setTarget:*anObject*

Sets the Executive's target to be *anObject*.

See also: **target**

action

– (SEL)**action**

Returns the action that is sent to the target when an asynchronous command completes.

See also: **setAction:**

setAction

– setAction:(SEL)*aSelector*

Sets the action that is sent to the target when an asynchronous command completes..

See also: **action**

period

– (double)**period**

Returns the period at which the timed entry executes when looking for completed commands.

See also: **setPeriod:**

setPeriod

– setPeriod:(double)*p*

Sets the period at which the timed entry executes when looking for completed commands.

See also: **period**

execute

– (int)**execute:**(const char *)*command*

Begins execution of *command* synchronously. Returns the result of the system(3) call.

See also: **execute:async:**, **execute:environs:async:**

execute:async:

– (int)**execute:**(const char *)*command* **async:**(BOOL)*async*

Begins execution of *command.* If *asnyc* is YES then the method returns immediately with the command identifier (a unique integer) that can be used when the command eventually notifies the caller that it has been completed. If *async* is NO then it returns the result of the system(3) call.

See also: **execute:**, **execute:environs:async:**

execute:environs:async:

– (int)**execute:**(const char *)*command* **environs:**(const char *)*environs* **async:**(BOOL)*async*

Begins execution of *command.* If *async* is YES then the method returns immediately with the command identifier (a unique integer) that can be used when the command eventually notifies the caller that it has been completed. If *async* is NO then it returns the result of the system(3) call. The *environs* argument contains command-line style environment variable definitions that are prepended to the command line before execution.

See also: **execute:**, **execute:async:**

pipe:to::

– (int)**pipe:**(const char *)*command* **to:***anObject* **:**(SEL)*aSelector* **async:**(BOOL)*async*

Opens a pipe to the command *command* and sends the output lines to *anObject* with the selector *aSelector*. *aSelector* should be a method that takes one argument, the line that is bring processed. See below under **pipe:environs:to::async:** for the semantics of an asynchronous request.

See also: **pipe:environs:to::async:**

pipe:environs:to::async:

– (int)**pipe:**(const char *)*command* **environs:**(const char *)*environs* **to:***anObject* **:**(SEL)*aSelector* **async:**(BOOL)*async*

Opens a pipe to the command *command* and sends the output lines to *anObject* with the selector *aSelector*. For synchronous commands, *aSelector* should be a method that takes one argument, the line that is being processed. For asynchronous commands, *aSelector* should be a method that takes two arguments, the command identifier that is returned by this method and the line that is being processed. The *environs* argument contains command-line style environment variable definitions that are prepended to the command line before execution. If **pipe:environs:to::async:** is called to operate asynchronously, the target of the Executive will be notified in the same manner as with **execute:environs:async:**.

See also: **pipe:to::async:**

showError

– **showError:**(int)*err*
– **showError:**(int)*err* **while:**(const char *)*doingWhat*
– **showError:**(int)*err* **while:**(const char *)*doingWhat* **on:**(const char *)*fname*
– **showError:**(int)*err* **while:**(const char *)*doingWhat* **on:**(const char *)*fname* **using:**(const char *)*prog*

Allows the application to have a regular set of error-reporting abilities in various degrees of granularity. The area in the upper portion of the NXRunAlertPanel panel will be the text (by named parameter) "*prog* error" The lower part will have the text "Error while *doingWhat fname* (error code *err*)"

All of the methods eventually call **showError:while:on:using:**. If any of the more general methods are called, they substitute defaults for the missing parameters.

The defaults for these methods are:

doingWhat	defaults to "executing a command"
fname	defaults to "on a file"
prog	defaults to "File"