NSTask

Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	Foundation/NSTask.h

Class Description

Using NSTask, your program can run another program as a subprocess and can monitor that program's execution. NSTask creates a separate executable entity; it differs from NSThread in that it does not share memory space with the process that creates it.

A task operates within an environment defined by the current values for several items: the current directory, standard input, standard output, standard error, and the values of any environment variables. By default, an NSTask inherits its environment from the process that launches it. If there are any values that should be different for the task, for example, if the current directory should change, you must change the value before you launch the task. A task's environment cannot be changed while it is running.

Creating and Launching an NSTask

There are two ways to create an NSTask. If it's sufficient for the task to run in the environment that it inherits from the process that creates it, use the class method **launchedTaskWithLaunchPath:arguments:**. This method both creates and executes (launches) the task. If you need to change the task's environment, create the task using **alloc** and **init**, use **set...** methods to change parts of the environment, then use the **launch** method to launch the task. For example, the following method runs tasks that take an input file and an output file as arguments. It reads these arguments, the task's executable, and the current directory from text fields before it launches the task:

```
- (void)runTask:(id)sender
{
    NSTask *aTask = [[NSTask alloc] init];
    NSMutableArray *args = [NSMutableArray array];

    /* set arguments */
    [args addObject:[[inputFile stringValue] lastPathComponent]];
    [args addObject:[outputFile stringValue]];
    [aTask setCurrentDirectoryPath:[[inputFile stringValue]
          stringByDeletingLastPathComponent]];
    [aTask setLaunchPath:[taskField stringValue]];
    [aTask setArguments:args];
    [aTask launch];
```

```
}
```

If you create an NSTask object in this manner, you must be sure to set the executable name using **setLaunchPath:**. If you don't, an NSInvalidArgumentException is raised.

Ending an NSTask

Normally, you want the task that you've launched to run to completion. When an NSTask exits, it posts an NSTaskDidTerminateNotification to the default notification center. You can add one of the custom objects in your program as an observer of the notification and check the task's exit status (using **terminationStatus**) in the observer method. For example:

```
-(id)init
{
    self = [super init];
    [[NSNotificationCenter defaultCenter] addObserver:self
      selector:@selector(checkATaskStatus:)
      name:NSTaskDidTerminateNotification
      object:nil];
    return self;
}

- (void)checkATaskStatus:(NSNotification *)aNotification
{
    int status = [[aNotification object] terminationStatus];
    if (status == ATASK_SUCCESS_VALUE)
      NSLog(@"Task succeeded.");
   else
      NSLog(@"Task failed.");
}
```

If you need to force a task to end execution, send **terminate** to the NSTask object.

Method Types

Creating and initializing an NSTask	+ launchedTaskWithLaunchPath:arguments: – init
Returning task information	– arguments – currentDirectoryPath – environment – launchPath – standardError – standardInput – standardOutput

Running and stopping an NSTask	– launch – terminate – waitUntilExit
Querying the NSTask state	– isRunning – terminationStatus
Setting up an NSTask	– setArguments: – setCurrentDirectoryPath: – setEnvironment: – setLaunchPath: – setStandardError: – setStandardInput: – setStandardOutput:

Class Methods

launchedTaskWithLaunchPath:arguments:

+ (NSTask *)**launchedTaskWithLaunchPath:**(NSString *)*path* **arguments:**(NSArray *)*arguments*

Creates and launches a task with the executable specified in *path*, providing the argument in the array *arguments*. The task inherits its environment from the process that invokes this method.

See also: – **init**

Instance Methods

arguments

– (NSArray *)**arguments**

Returns the command arguments that were used when the task was launched.

See also: – **setArguments:**

currentDirectoryPath

– (NSString *)**currentDirectoryPath**

Returns the task's current directory.

See also: – **setCurrentDirectoryPath:**

environment

– (NSDictionary *)**environment**

Returns a dictionary of variables for the environment from which the task was launched. The dictionary keys are the environment variable names.

See also: – **environment** (NSProcessInfo), – **setEnvironment:**

init

– (id)**init**

Returns an initialized NSTask object with the environment of the current process. Usually, if the current process's environment is sufficient, you use the class method **launchedTaskWithLaunchPath:arguments:** to create an run the task. Otherwise, you use **alloc** and **init** and then set up the environment before you launch the task.

isRunning

– (BOOL)**isRunning**

Returns YES if the NSTask is still running, otherwise NO. NO either means the task could not run or it has terminated.

See also: – **launch**, – **terminate**, – **waitUntilExit**

launch

– (void)**launch**

Launches the task represented by the NSTask object. Raises an NSInvalidArgumentException if the launch path has not been set or is invalid or if it fails to create a process.

See also: – **launchPath**, – **setLaunchPath:**, – **terminate**, – **waitUntilExit**

launchPath

– (NSString *)**launchPath**

Returns the path of the NSTask's executable.

See also: + **launchedTaskWithLaunchPath:arguments:**, – **setLaunchPath:**

setArguments:

– (void)**setArguments:**(NSArray *)*arguments*

Sets the command arguments that should be used to launch the path to *arguments*. If this method (or **launchedTaskWithLaunchPath:arguments:**) isn't used, the command is launched with no arguments. You cannot use this method if the task has already been launched. If you do, it raises an NSInvalidArgumentException.

See also: – **arguments**

setCurrentDirectoryPath:

– (void)**setCurrentDirectoryPath:**(NSString *)*path*

Sets the current directory for the task to *path*. If this method isn't used, the current directory is inherited from the process that created the NSTask. You cannot use this method if the task has already been launched. If you do, it raises an NSInvalidArgumentException.

See also: – **currentDirectoryPath**

setEnvironment:

– (void)**setEnvironment:**(NSDictionary *)*environmentDictionary*

Sets the environment for the task to *environmentDictionary.* The environment is a dictionary of environment variable values whose keys are the variable names. If this method isn't used, the environment is inherited from the process that created the NSTask. You cannot use this method if the task has already been launched. If you do, it raises an NSInvalidArgumentException.

See also: – **environment**

setLaunchPath:

– (void)**setLaunchPath:**(NSString *)*path*

Sets the task's executable to path. You must use this method before you send **launch** to launch the task or else use **launchedTaskWithLaunchPath:arguments:**. If you don't, NSTask raises an NSInvalidArgumentException.

See also: – **launchPath**

setStandardError:

– (void)**setStandardError:**(id)*file*

Sets standard error for the task to *file*, which can be either an NSFileHandle or an NSPipe object. If *file* is an NSPipe, launching the NSTask automatically closes the write end of the pipe in the current task. If you're using a pipe for standard error, use an NSPipe instance as the argument to this method. Don't create a handle for the pipe and pass that as the argument. If you do, the write end of the pipe won't be closed automatically.

If this method isn't used, the standard error is inherited from the process that created the NSTask. You cannot use this method if the task has already been launched. If you do, it raises an NSInvalidArgumentException.

See also: – **standardError**

setStandardInput:

– (void)**setStandardInput:**(id)*file*

Sets standard input for the task to *file*, which can be either an NSFileHandle or an NSPipe object. If *file* is an NSPipe, launching the NSTask automatically closes the read end of the pipe in the current task. If you're using a pipe for standard input, use an NSPipe instance as the argument to this method. Don't create a handle for the pipe and pass that as the argument. If you do, the read end of the pipe won't be closed automatically.

If this method isn't used, the standard input is inherited from the process that created the NSTask. You cannot use this method if the task has already been launched. If you do, it raises an NSInvalidArgumentException.

See also: – **standardInput**

setStandardOutput:

– (void)**setStandardOutput:**(id)*file*

Sets standard output for the task to *file*, which can be either an NSFileHandle or an NSPipe object. If *file* is an NSPipe, launching the NSTask automatically closes the write end of the pipe in the current task. If you're using a pipe for standard output, use an NSPipe instance as the argument to this method. Don't create a handle for the pipe and pass that as the argument. If you do, the write end of the pipe won't be closed automatically.

If this method isn't used, the current standard output is inherited from the process that created the NSTask. You cannot use this method if the task has already been launched. If you do, it raises an NSInvalidArgumentException.

See also: – **standardOutput**

standardError

– (id)**standardError**

Returns the standard error file used by the task. Standard error is where all diagnostic messages are sent. The object returned is either an NSFileHandle or an NSPipe instance, depending on what type of object was passed to the **setStandardError:** method.

See also: – **setStandardError:**

standardInput

– (id)**standardInput**

Returns the standard input file used by the task. Standard input is where the task takes its input from unless otherwise specified. The object returned is either an NSFileHandle or an NSPipe instance, depending on what type of object was passed to the **setStandardInput:** method.

See also: – **setStandardInput:**

standardOutput

– (id)**standardOutput**

Returns the standard output file used by the task. Standard output is where the task displays its output. The object returned is either an NSFileHandle or an NSPipe instance, depending on what type of object was passed to the **setStandardOutput:** method.

See also: – **setStandardOutput:**

terminate

– (void)**terminate**

Sends a terminate signal to the NSTask and all of its subtasks, posting a NSTaskDidTerminateNotification to the default notification center. This method has no effect if the NSTask was already launched and has already finished executing. If the task hasn't been launched yet, this method raises an NSInvalidArgumentException.

It is not always possible to terminate the task because the task might be ignoring the terminate signal.

See also: + **launchedTaskWithPath:arguments:**, – **launch**, – **terminationStatus**, – **waitUntilExit**

terminationStatus

– (int)**terminationStatus**

Returns the value returned by the task's executable. The return value indicates the exit status of the task. Each task defines and documents how its return value should be interpreted. (For example, many UNIX commands return 0 if they complete successfully or an error code if they don't. You'll need to look at the documentation for that task to learn what values it returns under what circumstances.) This method raises an NSInvalidArgumentException if the task is still running. Verify that the task is not running before you use it.

```
if (![aTask isRunning]) {
    int return = [aTask terminationStatus];
    if (status == ATASK_SUCCESS_VALUE)
       NSLog(@"Task succeeded.");
    else
       NSLog(@"Task failed.");
}
```

See also: – **terminate**, – **waitUntilExit**

waitUntilExit

– (void)**waitUntilExit**

Suspends your program until the tasks is finished. This method first checks to see if the task is still running using **isRunning**. Then it polls the current run loop using NSDefaultRunLoopMode until the task completes. (See the NSRunLoop class specification for more information on run loops and run loop modes.)

```
int return = [aTask terminationStatus];

[aTask launch];
[aTask waitUntilExit];
return = [aTask terminationStatus];
if (status == ATASK_SUCCESS_VALUE)
   NSLog(@"Task succeeded.");
else
   NSLog(@"Task failed.");
```

See also: – **launch**, – **terminate**

Notifications

NSTaskDidTerminateNotification

Posted when the task has stopped execution. This can be posted either when the task has exited normally or as a result of **terminate** being sent to the NSTask. The observer method can use **terminationStatus** to determine why the task died. See the class description for an example.

Notification Object	The NSTask that was terminated
Userinfo	None.