NSTask Class Reference

Cocoa > Process Management

2007-01-31

Contents

NSTask Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Companion guide	Interacting with the Operating System
Declared in	NSTask.h
Related sample code	Moriarity MP3 Player

Overview

Using the `NSTask` class, your program can run another program as a subprocess and can monitor that program's execution. An `NSTask` object 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` object 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.

An `NSTask` object can only be run once. Subsequent attempts to run the task raise an error.

Tasks

Creating and Initializing an NSTask Object

`+ launchedTaskWithLaunchPath:arguments:` (page 7)
: Creates and launches a task with a specified executable and arguments.

`- init` (page 9)
: Returns an initialized `NSTask` object with the environment of the current process.

Returning Task Information

- `arguments` (page 8)
  Returns the arguments used when the receiver was launched.
- `currentDirectoryPath` (page 8)
  Returns the task's current directory.
- `environment` (page 8)
  Returns a dictionary of variables for the environment from which the receiver was launched.
- `launchPath` (page 10)
  Returns the path of the receiver's executable.
- `processIdentifier` (page 11)
  Returns the receiver's process identifier.
- `standardError` (page 15)
  Returns the standard error file used by the receiver.
- `standardInput` (page 15)
  Returns the standard input file used by the receiver.
- `standardOutput` (page 15)
  Returns the standard output file used by the receiver.

Running and Stopping a Task

- `interrupt` (page 9)
  Sends an interrupt signal to the receiver and all of its subtasks.
- `launch` (page 10)
  Launches the task represented by the receiver.
- `resume` (page 11)
  Resumes execution of the receiver task that had previously been suspended with a `suspend` (page 16) message.
- `suspend` (page 16)
  Suspends execution of the receiver task.
- `terminate` (page 16)
  Sends a terminate signal to the receiver and all of its subtasks.
- `waitUntilExit` (page 17)
  Block until the receiver is finished.

Querying the Task State

- `isRunning` (page 10)
  Returns whether the receiver is still running.
- `terminationStatus` (page 17)
  Returns the exit status returned by the receiver's executable.

Configuring an NSTask Object

`- setArguments:` (page 11)
: Sets the command arguments that should be used to launch the executable.

`- setCurrentDirectoryPath:` (page 12)
: Sets the current directory for the receiver.

`- setEnvironment:` (page 12)
: Sets the environment for the receiver.

`- setLaunchPath:` (page 13)
: Sets the receiver's executable.

`- setStandardError:` (page 13)
: Sets the standard error for the receiver.

`- setStandardInput:` (page 14)
: Sets the standard input for the receiver.

`- setStandardOutput:` (page 14)
: Sets the standard output for the receiver.

Class Methods

launchedTaskWithLaunchPath:arguments:

Creates and launches a task with a specified executable and arguments.

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

Parameters

path
: The path to the executable.

arguments
: An array of `NSString` objects that supplies the arguments to the task. If *arguments* is `nil`, an `NSInvalidArgumentException` is raised.

Discussion
The task inherits its environment from the process that invokes this method.

The `NSTask` object converts both *path* and the strings in *arguments* to appropriate C-style strings (using `fileSystemRepresentation`) before passing them to the task via `argv[]` . The strings in *arguments* do not undergo shell expansion, so you do not need to do special quoting, and shell variables, such as `$PWD`, are not resolved.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- init` (page 9)

Declared In
`NSTask.h`

Instance Methods

arguments

Returns the arguments used when the receiver was launched.

```
- (NSArray *)arguments
```

Return Value
An array of `NSString` objects containing the arguments used when the receiver was launched.

Availability
Available in Mac OS X v10.0 and later.

See Also
`– setArguments:` (page 11)

Declared In
`NSTask.h`

currentDirectoryPath

Returns the task's current directory.

```
- (NSString *)currentDirectoryPath
```

Return Value
The task's current working directory.

Availability
Available in Mac OS X v10.0 and later.

See Also
`– setCurrentDirectoryPath:` (page 12)

Declared In
`NSTask.h`

environment

Returns a dictionary of variables for the environment from which the receiver was launched.

```
- (NSDictionary *)environment
```

Return Value
A dictionary of variables for the environment from which the receiver was launched. The dictionary keys are the environment variable names.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `- setEnvironment:` (page 12)
- `- environment` (NSProcessInfo)

Declared In
`NSTask.h`

init

Returns an initialized `NSTask` object with the environment of the current process.

```
- (id)init
```

Return Value
An initialized `NSTask` object with the environment of the current process.

Discussion
If you need to modify the environment of a task, use alloc and init, and then set up the environment before launching the new task. Otherwise, just use the class method `launchedTaskWithLaunchPath:arguments:` (page 7) to create and run the task.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSTask.h`

interrupt

Sends an interrupt signal to the receiver and all of its subtasks.

```
- (void)interrupt
```

Discussion
If the task terminates as a result, which is the default behavior, an `NSTaskDidTerminateNotification` (page 18) gets sent to the default notification center. This method has no effect if the receiver was already launched and has already finished executing. If the receiver has not been launched yet, this method raises an `NSInvalidArgumentException`.

It is not always possible to interrupt the receiver because it might be ignoring the interrupt signal. `interrupt` sends `SIGINT`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSTask.h`

isRunning

Returns whether the receiver is still running.

```
- (BOOL)isRunning
```

Return Value
`YES` if the receiver is still running, otherwise `NO`. `NO` means either the receiver could not run or it has terminated.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `launch` (page 10)
- `terminate` (page 16)
- `waitUntilExit` (page 17)

Declared In
`NSTask.h`

launch

Launches the task represented by the receiver.

```
- (void)launch
```

Discussion
Raises an `NSInvalidArgumentException` if the launch path has not been set or is invalid or if it fails to create a process.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `launchPath` (page 10)
- `setLaunchPath:` (page 13)
- `terminate` (page 16)
- `waitUntilExit` (page 17)

Declared In
`NSTask.h`

launchPath

Returns the path of the receiver's executable.

```
- (NSString *)launchPath
```

Return Value
The path of the receiver's executable.

Availability
Available in Mac OS X v10.0 and later.

See Also

`+ launchedTaskWithLaunchPath:arguments:` (page 7)

`- setLaunchPath:` (page 13)

Declared In

`NSTask.h`

processIdentifier

Returns the receiver’s process identifier.

```
- (int)processIdentifier
```

Return Value

The receiver’s process identifier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSTask.h`

resume

Resumes execution of the receiver task that had previously been suspended with a `suspend` (page 16) message.

```
- (BOOL)resume
```

Return Value

`YES` if the receiver was able to resume execution, `NO` otherwise.

Discussion

If multiple `suspend` messages were sent to the receiver, an equal number of `resume` messages must be sent before the task resumes execution.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSTask.h`

setArguments:

Sets the command arguments that should be used to launch the executable.

```
- (void)setArguments:(NSArray *)arguments
```

Parameters

`arguments`

An array of `NSString` objects that supplies the arguments to the task. If *`arguments`* is `nil`, an `NSInvalidArgumentException` is raised.

Discussion
The `NSTask` object converts both *`path`* and the strings in *`arguments`* to appropriate C-style strings (using `fileSystemRepresentation`) before passing them to the task via `argv[]` . The strings in *`arguments`* do not undergo shell expansion, so you do not need to do special quoting, and shell variables, such as `$PWD`, are not resolved.

Availability
Available in Mac OS X v10.0 and later.

See Also
`– arguments` (page 8)

Declared In
`NSTask.h`

setCurrentDirectoryPath:

Sets the current directory for the receiver.

```
- (void)setCurrentDirectoryPath:(NSString *)path
```

Parameters

`path`
The current directory for the task.

Discussion
If this method isn't used, the current directory is inherited from the process that created the receiver. This method raises an `NSInvalidArgumentException` if the receiver has already been launched.

Availability
Available in Mac OS X v10.0 and later.

See Also
`– currentDirectoryPath` (page 8)

Declared In
`NSTask.h`

setEnvironment:

Sets the environment for the receiver.

```
- (void)setEnvironment:(NSDictionary *)environmentDictionary
```

Parameters

`environmentDictionary`
A dictionary of environment variable values whose keys are the variable names.

Discussion
If this method isn't used, the environment is inherited from the process that created the receiver. This method raises an `NSInvalidArgumentException` if the receiver has already been launched.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `environment` (page 8)

Declared In
`NSTask.h`

setLaunchPath:

Sets the receiver's executable.

```
- (void)setLaunchPath:(NSString *)path
```

Parameters

`path`
The path to the executable.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `launchPath` (page 10)

Declared In
`NSTask.h`

setStandardError:

Sets the standard error for the receiver.

```
- (void)setStandardError:(id)file
```

Parameters

`file`
The standard error for the receiver, which can be either an `NSFileHandle` or an `NSPipe` object.

Discussion
If *`file`* is an `NSPipe` object, launching the receiver automatically closes the write end of the pipe in the current task. Don't create a handle for the pipe and pass that as the argument, or 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 receiver. This method raises an `NSInvalidArgumentException` if the receiver has already been launched.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `standardError` (page 15)

Declared In
`NSTask.h`

setStandardInput:

Sets the standard input for the receiver.

```
- (void)setStandardInput:(id)file
```

Parameters

`file`

The standard input for the receiver, which can be either an `NSFileHandle` or an `NSPipe` object.

Discussion

If *`file`* is an `NSPipe` object, launching the receiver automatically closes the read end of the pipe in the current task. Don't create a handle for the pipe and pass that as the argument, or 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 receiver. This method raises an `NSInvalidArgumentException` if the receiver has already been launched.

Availability

Available in Mac OS X v10.0 and later.

See Also

`– standardInput` (page 15)

Declared In

`NSTask.h`

setStandardOutput:

Sets the standard output for the receiver.

```
- (void)setStandardOutput:(id)file
```

Parameters

`file`

The standard output for the receiver, which can be either an `NSFileHandle` or an `NSPipe` object.

Discussion

If *`file`* is an `NSPipe` object, launching the receiver automatically closes the write end of the pipe in the current task. Don't create a handle for the pipe and pass that as the argument, or the write end of the pipe won't be closed automatically.

If this method isn't used, the standard output is inherited from the process that created the receiver. This method raises an `NSInvalidArgumentException` if the receiver has already been launched.

Availability

Available in Mac OS X v10.0 and later.

See Also

`– standardOutput` (page 15)

Declared In

`NSTask.h`

standardError

Returns the standard error file used by the receiver.

```
- (id)standardError
```

Return Value
The standard error file used by the receiver.

Discussion
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 `setStandardError:` (page 13).

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setStandardError:` (page 13)

Declared In
`NSTask.h`

standardInput

Returns the standard input file used by the receiver.

```
- (id)standardInput
```

Return Value
The standard input file used by the receiver.

Discussion
Standard input is where the receiver 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:` (page 14) method.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setStandardInput:` (page 14)

Declared In
`NSTask.h`

standardOutput

Returns the standard output file used by the receiver.

```
- (id)standardOutput
```

Return Value
The standard output file used by the receiver.

Discussion
Standard output is where the receiver 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:` (page 14) method.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setStandardOutput:` (page 14)

Declared In
`NSTask.h`

suspend

Suspends execution of the receiver task.

```
- (BOOL)suspend
```

Return Value
`YES` if the receiver was successfully suspended, `NO` otherwise.

Discussion
Multiple `suspend` messages can be sent, but they must be balanced with an equal number of `resume` (page 11) messages before the task resumes execution.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSTask.h`

terminate

Sends a terminate signal to the receiver and all of its subtasks.

```
- (void)terminate
```

Discussion
If the task terminates as a result, which is the default behavior, an `NSTaskDidTerminateNotification` (page 18) gets sent to the default notification center. This method has no effect if the receiver was already launched and has already finished executing. If the receiver has not been launched yet, this method raises an `NSInvalidArgumentException`.

It is not always possible to terminate the receiver because it might be ignoring the terminate signal. `terminate` sends `SIGTERM`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ launchedTaskWithLaunchPath:arguments:` (page 7)

- `launch` (page 10)
- `terminationStatus` (page 17)
- `waitUntilExit` (page 17)

Declared In
`NSTask.h`

terminationStatus

Returns the exit status returned by the receiver's executable.

```
- (int)terminationStatus
```

Return Value
The exit status returned by the receiver's executable.

Discussion
Each task defines and documents how its return value should be interpreted. For example, many 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 receiver is still running. Verify that the receiver is not running before you use it.

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

Availability
Available in Mac OS X v10.0 and later.

See Also
- `terminate` (page 16)
- `waitUntilExit` (page 17)

Declared In
`NSTask.h`

waitUntilExit

Block until the receiver is finished.

```
- (void)waitUntilExit
```

Discussion
This method first checks to see if the receiver is still running using `isRunning` (page 10). Then it polls the current run loop using `NSDefaultRunLoopMode` until the task completes.

```
[aTask launch];
```

```
[aTask waitUntilExit];
int status = [aTask terminationStatus];

if (status == ATASK_SUCCESS_VALUE)
    NSLog(@"Task succeeded.");
else
    NSLog(@"Task failed.");
```

Availability
Available in Mac OS X v10.0 and later.

See Also
- `launch` (page 10)
- `terminate` (page 16)

Declared In
`NSTask.h`

Notifications

NSTaskDidTerminateNotification

Posted when the task has stopped execution. This notification can be posted either when the task has exited normally or as a result of `terminate` (page 16) being sent to the `NSTask` object. If the `NSTask` object gets released, however, this notification will not get sent, as the port the message would have been sent on was released as part of the task release. The observer method can use `terminationStatus` (page 17) to determine why the task died. See "Ending an NSTask" for an example.

The notification object is the `NSTask` object that was terminated. This notification does not contain a *`userInfo`* dictionary.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSTask.h`

Document Revision History

This table describes the changes to *NSTask Class Reference*.

Date	Notes
2007-01-31	Corrected the description for the setStandardInput: method. Updated for Mac OS X v10.5.
2006-05-23	First publication of this content as a separate document.

Index