for Release 2.0 written by C. L. Oei, NeXT Computer, Inc.

Subprocess

INHERITS FROM	Object
REQUIRES HEADER FILES	Subprocess.h
DEFINED IN	NextDeveloper/Examples/Subprocess, release 2.0

CLASS DESCRIPTION

Subprocess facilitates the management of concurrent, asynchronous UNIX processes within a NeXTstep application. Methods are provided for the creation, termination and communication with the underlying UNIX process. The UNIX subprocess communicates with its parent NeXTstep application through delegation. Three delegate methods provide notification of pending output, termination and errors from the UNIX subprocess. It is the responsibility of the Subprocess instantiator to implement the three delegate methods and decide what to do with the resulting data. In addition to providing a controlled NeXTstep interface to standard UNIX utilities (i.e., `ls, find, man, rdist`), the Subprocess can also provide, on request, the environment necessary for UNIX processes requiring pseudo terminal (or *pty*) support. Some UNIX applications that require pty support include `ftp, gdb, sh, csh, kermit,` and `tip`.

INSTANCE VARIABLES

Inherited from Object	Class	isa;
Declared in Subprocess	id	delegate;
	FILE	*fpToChild;
	int	fromChild;
	int	childPid;

delegate	The object that receives notification messages from the Subprocess.
fpToChild	File pointer to the standard input of the child subprocess.
fromChild	File descriptor from the standard output of the child subprocess.
childPid	The process id number of the child subprocess.

METHOD TYPES

Initializing a Subprocess	- init: - init:withDelegate:andPtySupport:andStdError:
Terminating a Subprocess	- terminate:
Sending Data to a Subprocess	- send: - send:withNewline: - terminateInput
Assigning a Delegate	- setDelegate:

- delegate

Messages Implemented by the Delegate

- subprocessDone
- subprocessError:
- subprocessOutput:

INSTANCE METHODS

init:

- **init:**(const char *)*subprocessString*

Spawns the subprocess as specified in *subprocessString*. This method applies the **init:withDelegate:andPtySupport:andStdError:** method with no delegate, no pseudo terminal support, and requests that standard error for the subprocess be returned with the standard output buffer.

See also: - **init:withDelegate:andPtySupport:andStdError:,** ± **terminate:**

init:withDelegate:andPtySupport:andStdError:

- **init:**(const char *)*subprocessString*
  withDelegate:*theDelegate*
  andPtySupport:(BOOL)*wantsPty*
  andStdErr:(BOOL)*wantsStdErr*;

Spawns the subprocess as specified in *subprocessString* as a separate UNIX process and

attaches the subprocess' standard input and standard output to the Subprocess instance for future operations. If *wantsStdErr* is YES, then the subprocess' standard error will be returned with the standard output buffer. Set *wantsPty* to YES if the UNIX subprocess requires pseudo terminal support (see the UNIX manual page *pty(4)* for more information on pseudo terminals). *TheDelegate* should be able to respond to any of the three methods described below.

See also: **- init:,** ± **terminate:**

INSTANCE METHODS

delegate

- delegate

Returns the Subprocess object's delegate.

See also: ± **setDelegate:**

send:withNewline:

- (BOOL)**send:**(const char *)*string* **withNewline:**(BOOL)*wantNewline*

Sends *string* to the UNIX subprocess. If *wantNewline* is YES, a newline is also sent to the subprocess.

See also: ± **send:**

send:

- **send:**(const char *)*string*

Sends *string* to the UNIX subprocess automatically appending a newline. This method applies the **send:withNewline:** method.

See also: ± **send:withNewline:**

setDelegate:

- **setDelegate:***anObject*

Makes *anObject* the Subprocess' delegate. The delegate should be able to (but is not necessarily required to) respond to the messages **subprocessDone**, **subprocessError:**, and **subprocessOutput:**. See methods implemented by the delegate below.

See also: ± **delegate:**

terminate:

- **terminate:***sender*

Forces the subprocess to terminate gracefully. Closes all communication connections to the subprocess and sends a terminate signal (SIGTERM) to the subprocess. Sending this message implies sending the **terminateInput** message.

You should terminate a subprocess instance before your application terminates. One way would be to override the Application object's delegate **appWillTerminate:** method with an intervening message to terminate the subprocess object.

Sending **terminate:** multiple times will not cause undesirable effects.

See also: ± **terminateInput, subprocessDone:**

terminateInput

- **terminateInput**

Closes the standard input communication connection to the subprocess, which effectively sends an end-of-file (EOF) to the subprocess.

METHODS IMPLEMENTED BY THE DELEGATE

subprocessDone:

- **subprocessDone**

Sent to the delegate, if any, when the subprocess has terminated. You will have to decide whether termination of the subprocess warrants the termination of your application. Implies that the actions of sending a **terminate:** message has completed.

See also: ± **terminate:**

subprocessError:

- **subprocessOutput:**(const char *)*errorString*;

Sent to the delegate, if any, when a fatal error occurs during the management of the subprocess. If a fatal error occurs, it is usually during subprocess creation. It is up to the delegate to decide if an error warrants termination of the application. Possible errors

include, but are not limited to

± "Error grabbing ptys for subprocess."
± "Error starting UNIX pipes to subprocess."
± "Error starting UNIX vfork of subprocess."

subprocessOutput:

- **subprocessOutput:**(char *)*buffer*;

Sent to the delegate, if any, when there is output data available from the subprocess. *Buffer* is only valid until the next time a **subprocessOutput:** message is sent, so make a copy of *buffer* if future processing is necessary. You should choose carefully when deciding whether or not to send a **send:** message to the subprocess in this delegate method implementation. Sending a message may create a deadlock situation in your application.