MiscSubprocess

Inherits From: Object

Declared In: <misckit/MiscSubprocess.h>

Class Description

No Description.

Instance Variables

id **delegate**;
id **environment**;
FILE ***fpToChild**;
FILE ***fpFromChild**;
int **stdoutFromChild**;
int **stderrFromChild**;
int **childPid**;

id **stdoutBuffer**;
id **stderrBuffer**;
BOOL **paused**;
BOOL **running**;

delegate	MiscSubprocess' delegate object.
environment	MiscStringArray that holds the child's environment.
fpToChild	Pipe for sending data to the child process' stdin.
fpFromChild	Pipe for receiving data from the child process' stdout.
stdoutFromChild	File descriptor of pipe for receiving data from the child process' stdout.
stderrFromChild	File descriptor of pipe for receiving data from the child process' stderr.
childPid	PID of the child process.
stdoutBuffer	MiscString used to buffer the child's stdout.
stderrBuffer	MiscString used to buffer the child's stderr.
paused	True if child process is paused.
running	True if the child process is running.

Method Types

Initializing a MiscSubProcess:	- init - init: - init:withDelegate:

Sending Data to Child:	- send: - send:withNewline: - terminateInput
Sending Data From Child to Delegate:	- flushBuffer: - flushBuffer:as: - outputMethodForBuffer:
Obtaining Information About Child:	- environment - isPaused - isRunning - pid
Controlling Child Process:	- execChild: - execute: - pause: - resume: - terminate:
Delegate methods:	- delegate - setDelegate: - subprocess:done:: - subprocess:error: - subprocess:output: - subprocess:stderrOutput:

Instance Methods

delegate

- **delegate**

Returns the MiscSubprocess' delegate object, the object responsible for handling errors and receiving output from the child process.

See also: ±**setDelegate:** and the delegate methods at the end of this document.

environment

- **environment**

Returns a MiscStringArray object that contains the environment that will be set up for the child process when it is started. If you wish to alter the environment, add to or modify the MiscStringArray returned by this method. Under no circumstances should you free the object returned to you; it is owned by the MiscSubprocess instance.

See also: ±**execute:**

execChild:

- **execChild:**(const char *)*aString*

This is used internally by MiscSubprocess to actually start the child process, using the UNIX execle(). If you wish to override how a child process is started, you should override this method and not ±**execute:**.

See also: ±**environment** and ±**execute:**

execute:

- **execute:**(const char *)*aString*

Sets up pipes to a child process and starts it executing the command in *aString*. Returns *self*.

See also: ±**environment**, ±**execChild:**, ±**pause:**, ±**resume:**, and ±**terminate:**

flushBuffer:

- **flushBuffer:***aString*

Flushes the buffer *aString* to the delegate. This is used internally by MiscSubprocess to send the stdin and stdout of the child process to the delegate.

See also: ±flushBuffer:as:, **±outputMethodForBuffer:**

flushBuffer:as:

- **flushBuffer:***aString*
  as:*aBuffer*

This method is used internally by the MiscSubprocess object to flush the stderr and stdout buffers of the child process. The string value of *aString* will be passed on to the delegate via the ±**subprocess:output:** or ±**subprocess:stderrOutput:** delegate methods. This method could be used to spoof data as if it came from the child process. Returns *self*.

See also: ±flushBuffer:, **±outputMethodForBuffer:**

init

- **init**

Initializes a new instance of MiscSubprocess. Returns *self*.

See also: ±init: and ±**init:withDelegate:**

init:

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

Initializes a new instance of MiscSubprocess and runs the command in *aString* with no delegate object. If *aString* is NULL, no command is executed. Returns *self*.

See also: ±**execute:**, ±**init**, and ±**init:withDelegate:**

init:withDelegate:

- **init:**(const char *)*aString*
  withDelegate:*theDelegate*

This is the designated initializer for the MiscSubProcess class. Initializes a new instance of MiscSubprocess and runs the command in *aString* with *theDelegate* as the delegate object. If *aString* is NULL, then no command is run; the instance variables should be set up as required and then ±**execute:** should be called. Returns *self*.

See also: ±**execute:**, ±**init**, and ±**init:**

isPaused

- (BOOL)**isPaused**

Returns YES is the child process is paused. Returns NO otherwise.

See also: ±**isRunning**, ±**pause:**, and ±**resume:**

isRunning

- (BOOL)**isRunning**

Returns YES if the child process is running, NO otherwise.

See also: ±**isPaused**, ±**pause:**, and ±**resume:**

outputMethodForBuffer:

- (SEL)**outputMethodForBuffer:***aBuffer*

Used internally by the MiscSubprocess object to determine which delegate method to send for *aBuffer*. It will return the selector for either ±**subprocess:output:** or ±**subprocess:stderrOutput:** depending on whether the argument is the buffer being used for stderr or stdout of the child process. Returns NULL if the argument is some other buffer.

See also: **±flushBuffer:** and **±flushBuffer:as:**

pause:

- **pause:***sender*

Pause the child process by sending it a SIGSTOP signal. Returns *self*.

See also: **±isPaused**, ±**isRunning**, ±**resume:**, and ±**terminate:**

pid

- (int)**pid**

Returns the process ID (pid) of the child process, if it exists. If there is no child process running, the return value is undefined and invalid.

resume:

- **resume:***sender*

Resumes a paused child process by sending a SIGCONT signal. Returns *self*.

See also: **±isPaused**, ±**isRunning**, ±**pause:**, and ±**terminate:**

send:

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

Sends *string*, followed by a newline character (`\n') to the child process. Returns *self*.

See also: **±send:withNewLine:** and ±**terminateInput**

send:withNewline:

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

Sends *string* to the child process. If *wantNewLine* is true, then a newline character (`\n') is appended to *string*. Returns *self*.

See also: **±send:** and ±**terminateInput**

setDelegate:

- **setDelegate:***anObject*

Sets the delegate object. See the delegate methods below for a description of what information is sent to the delegate. Returns *self*.

See also: **±delegate**

terminate:

- **terminate:***sender*

Sends a SIGKILL to the child process, terminating it's execution. Returns *self*.

See also: **±pause:**, ±**resume:**, and ±**terminate:**

terminateInput

- **terminateInput**

Terminate the data being sent to the child process. This will send an EOF to the child's stdin. Returns *self*.

See also: **±send:** and **±send:withNewLine:**

Delegate Methods

subprocess:done::
- **subprocess:***sender* **done:**(int)*status* **:**(MiscSubprocessEndCode)*code*

Sent to the delegate when the child process completes. The exit code *code* is the reason for the process' termination, and is one of Misc_Exited, Misc_Stopped, Misc_Signaled, or Misc_UnknownEndCode. If the process exited normally, the exit code is returned in *status*. If the process was stopped, then *status* contains the number of the signal that caused the process to stop. If the process was signaled, then *status* contains the number of the signal that caused the process to terminate.

subprocess:output:
- **subprocess:***sender* **output:**(const char *)*buffer*

Sent whenever there is data on the process' standard output pipe. The data is passed in *buffer* and is only valid until the next method call and should therefore be cached locally if need be.

See also: **±subprocess:stderrOutput:**

subprocess:stderrOutput:
- **subprocess:***sender* **stderrOutput:**(const char *)*buffer*

Sent whenever there is data on the process' standard output pipe. The data is passed in *buffer* and is only valid until the next method call and should therefore be cached locally if need be.

See also: **±subprocess:output:**

subprocess:error:

- **subprocess:***sender* **error:**(const char *)*errorString*

Sent if an error occurs when dealing with the child process. Most errors will occur when trying to start the process. By default, this method is a part of a category of the object class which simply passes *errorString* to perror(). Override this method if you wish to attempt error recovery or present the error to the user somewhere besides the console.