MiscShell

Inherits From: Object

Declared In: misckit/MiscShell.h

Class Description

A MiscShell object combines a MiscSubprocess object (which executes a Unix process) with a MiscStringArray object (which holds the individual lines of output), and features a target/action paradigm so that messages can be sent to other objects for each line of process output.

NOTE: *If you want to load the MiscShell palette, you must load the MiscString palette first. The MiscShell object needs to use the MiscString class, and expects to find it already loaded by Interface Builder.*

You can use a MiscShell on its own to capture the output of a Unix command into a MiscStringArray. In addition, MiscShell includes a variety of convenience methods to facilitate hooking up MiscShell objects from

within Interface Builder. The MiscShell palette provides a MiscShell object with an inspector that can be used to type up a shell script for the MiscShell to execute, and allows you to prepare a script, hook up target/action methods, indicate where the shell's standard output should go and so on, all totally inside of Interface Builder. You can build some interesting
programs that do Unix-y file manipulation type things in this way, without writing any Objective-C code at all.

MiscShell operates more or less like this when told to execute a particular Unix command or shell script.

· Collect the values of various UI objects it knows about, and add their values to the subprocess's environment;
· Create a MiscSubprocess to execute the script;
· for each line of output from the script
  send a target/action message
  display that line of output on a UI object
  add that line of output to a list

Running a command and examining its output

The idea is that you allocate and initialize a MiscShell object, tell it what script to execute, and tell it to execute that script. The MiscShell collects the output generated by the script into individual lines and keeps these lines in a MiscStringArray.

A convenience method **initWithCommand:** is available to simplify the execution of some commands. You give this method a command you would like to have executed, and it executes it and returns when the command has completed. For instance, suppose you want to run the **who** command to count the number of users logged into your machine. You might use it like this:

```
id myShell = [[MiscShell alloc] initWithCommand:"who"];
printf("There are %d users\n", [myShell lineCount];
```

and you could loop through the individual lines of output with the **line**: method, which returns a MiscString object containing the contents of line i:

```
for ( i = 0; i < [myShell lineCount]; i++  ) {
    printf("Line %d: %s\n", i, [[myShell line:i] stringValueAndFree] );
```

(You must free the object returned by the **line**: method yourself.)   You can also use the **line:field:** method to extract a particular field of a particular line:

```
/* Username is the first field of the "who" output */
printf("First User: %s\n", [[myShell line:0 field:0] stringValueAndFree];
```

MiscShell is using a **MiscStringArray** internally to store the individual lines of script output; you can get access to this array directly with the **lines** method.

Displaying Standard Output on User Interface objects

With Interface Builder, you can drag a MiscShell off of its palette and set some of its instance variables to other objects in your user interface.   You can use MiscShell's inspector to type up a simple Unix script, or you can arrange for the script itself to be taken from the title, altTitle or stringValue of other objects in the interface This gives you an easy way, without writing any Objective-C,   to run a Unix command and display its output.

A MiscShell object can be connected to display its standard output on a variety of different user interface objects

· Text Fields
· Scrolling Text objects
· One-column NXBrowsers
· DBTableView or NXTableView, with the output separated into columns
· Anything else that understands **setStringValue**

by connecting the object's **standardOutput** outlet to a suitable user interface object.   MiscShell knows how to write data to various different UI objects.

If standardOutput is a ...	each line of output is ...
TextField	Written to the text field, replacing the previous contents
Scrolling Text	Written to the text, appended after the previous contents
NXBrowser	Appended as another row to the (single-column) browser. (The MiscShell automatically sets itself to be the delegate of the browser.)
DBTableView / NXTableView	Split into fields and appended to the table view. (The MiscShell automatically sets itself to be the data source of the table view.)
Another **MiscShell**	Written to the shell's standard input. *(Untested.)*

Each line of data written to a Table View is split into fields according to the **separator** instance variable, and individual fields will appear in individual columns of the table view.

In addition, the MiscShell supports the standard **target/action** paradigm.   Each time the MiscShell reads a complete line of output from the MiscSubprocess, it will execute the desired target/action message.

By default, MiscShell's **executeScript:** method will execute a command asynchronously, meaning that the executeScript: method will return right away, and the MiscShell will keep an eye on data being written by the subprocess and will wake up and handle it whenever necessary. This is OK for a long-running command, but sometimes you would like the shell just to read all the data and return from the executeScript: method when the script has completed. You can do this by setting the **runToCompletion** variable.

Taking Values from user interface objects.

You can also connect four instance variables **v1, v2, v3, v4** to other UI objects in your program. Before the script executes, MiscShell checks the stringValue of each of these objects and adds "varname=stringValue" statements to the script's Unix environment. For instance, if you have connected v1 to a slider, and then execute the script, the script will be able to check the **$v1** environment variable to find the current value of the slider.

If the variable is connected to a DBTableView, MiscShell will figure out the value of the currently selected rows of the table view and use that as the value of the variable (with columns separated by tabs, and rows by newlines.)

The MiscShell also adds a few other special convenience environment variables that your script can consult.

$**sender**	The string value of the object that sent the **execute*** method.
$**senderTag**	The tag of the object that sent the **execute*** method.
$**mainBundle**	The pathname of the application's mainBundle.

Magic Output Strings

Certain magic strings in the subprocess's output will cause MiscShell to take special actions. If your subprocess outputs the string `ALERT;`*a message here*`;Yes;No` MiscShell will split up the line on the semicolons (which must be included) and pass the arguments to **NXRunAlertPanel()** , and write the (integer) result of the Alert panel back to the shell's input. So your shell could read that line of input and react accordingly.

```
echo 'ALERT;Are you sure you want to reformat the disk?;Yes;No'
# MiscShell now runs an alert panel and writes the result
# of NXRunAlertPanel to this process.  We'll read 1 for Yes,
# 0 for No.
read ans
case $ans in
1)    # yes
      reformat-disk ;;
*)    # no
      ;;
esac
```

The string `OPEN` causes MiscShell to pop up an Open panel to prompt for a filename. The file chosen will then be written back to the subprocess.

The string *var*=*value*, where *var* is one of v1, v2, ... v9, will cause the message [*var* setStringValue:*value*] to be sent. This gives the shell script a way to write specific values to specific user interface objects. For instance, you could connect v2 to some text field, and during execution your script could do

```
echo v2=`date`
```

which would display the current date on that text field.

If MiscShell recognizes one of these special lines, it performs the appropriate action but does NOT do the normal target/action and standard output handling that would otherwise happen.

DBTableView and NXTableView

This section isn't written yet.   It will describe how you can make a MiscShell a delegate of a DBTableView (which it sets up automatically if you make the table view the MiscShell's standard output), and have the MiscShell resort its output in response to column moves in the table view.   Go try it.

Instance Variables

id **standardOutput**;
id **standardError**;
id **standardInput**;
id **v1**;
id **v2**;
id **v3**;
id **v4**;
SEL **action**;
MiscString ***script**;
MyMiscSubprocess ***process**;
MiscString ***currentLine**;
MiscString ***fullOutput**;
BOOL **currentLineIsComplete**;

BOOL **executionInProgress**;
BOOL **runToCompletion**;
MiscString ***execArg0**;
MiscString ***execArg1**;
MiscString ***execArg2**;
MiscStringArray ***lines**;
char **delimiter**;

standardOutput	An object to which the script's standard output should be sent
standardError	An object to which the script's standard error should be sent.
standardInput	Not currently implemented.
v1, v2, v3, v4	Objects whose string values can be interpolated into the script as it executes.
action	The message sent to the shell's target.
script	A MiscString containing the actual shell script to be executed.
process	The MiscSubprocess that's created to execute the script.
currentLine	The current output line from the script.
fullOutput	A MiscString containing all the output generated by the script.
currentLineIsComplete	Indicates whether the currentLine string is complete - i.e. includes output up to a newline.
executionInProgress	If true, the subprocess is currently executing.

runToCompletion	If true, the **executeScript:** method should run the script and wait for it to complete, rather than launching it and returning.
execArg0	No description.
execArg1	No description.
execArg2	No description.
lines	A MiscStringArray containing the individual output lines generated by the script.
delimiter	A field separator character, used to split individual output lines into fields. If 0, lines will be split on word boundaries (whitespace).

Method Types

- action
- delimiter
- doubleValue
- executeFromAltTitle:
- executeFromStringValue:
- executeFromTitle:
- executeScript:
- fieldsInLine:
- floatValue
- free
- init

- initWithCommand:
- intValue
- lines
- line:
- line:field:
- pause:
- read:
- resume:
- runToCompletion
- script
- setAction:
- setDelimiter:
- setExecArgs:::
- setRunToCompletion:
- setScript:
- setTarget:
- stringValue
- target
- takeStdinFrom:
- terminate:
- write:

Instance Methods

action

- (SEL)**action**

Returns the action message sent to the script's target whenever a complete line has been read.

See also:

delimiter

- (char)**delimiter**

Returns the character used to split output lines into fields. If it's 0, output lines will be split on word (whitespace) boundaries.

See also:

doubleValue

- (double)**doubleValue**

Equivalent to **atof([self stringValue])**.

See also:

executeFromAltTitle:

- **executeFromAltTitle:***sender*

Asks the sender for its **altTitle**, sets its script to that string, and executes the script. You might set up a button to send this message to the MiscShell, and give the button a Unix command as its alternate title (which can be set in the Button inspector in Interface Builder).

See also:

executeFromStringValue:

- **executeFromStringValue:***sender*

Sets the script to be the sender's **stringValue**, and executes the script.

See also:

executeFromTitle:

- **executeFromTitle:***sender*

Sets the script to be the sender's title, and executes the script.

See also:

executeScript:

- **executeScript:***sender*

Executes the script, which should have been set by one of the other methods..

See also:

fieldsInLine:

- (int)**fieldsInLine:***(int)*lineNum

Returns the number of fields in output line *lineNum*

See also:

floatValue

- (float)**floatValue**

Equivalent to **atof([self stringValue])** .

See also:

free

- **free**

Terminates the subprocess, frees its allocated instance variables and itself.

See also:

init

- **init**

Initializes a newly allocated MiscShell.   Sets the script to an empty MiscString, sets delimiter to '\0', sets runToCompletion to NO, and returns.   You'll need to set the script somehow after using init.

See also:

initWithCommand:

- **initWithCommand:**(const char *)*command*

A convenience method that initializes a newly allocated MiscShell, sets the script to the indicated command, sets runToCompletion to YES, executes the script, and waits for it to complete before returning.

See also:

intValue

- (int)**intValue**

Equivalent to **atoi([self stringValue])**.

See also:

lines

- (MiscStringArray *) **lines**

Returns the MiscStringArray object that the MiscShell is using to store individual lines of the process's output. You're on your honour not to muck around with it.

See also:

line:

- (MiscString *) **line:**(int)*lineNum*

Returns a newly allocated MiscString object containing the contents of output line *lineNum.* You should free this string yourself when you're done with it. The first line is line 0.

See also:

line:field:

- (MiscString *) **line:**(int)*lineNum* **field:**(int)*fieldNumber*

Returns a newly allocated MiscString object containing the contents of field number *fieldNumber* of output line *lineNum.* The appropriate output line is split into fields according to the **delimiter** variable. If it's 0, fields are split into whitespace-delimited words. The first field is field 0.

See also:

lines

- (MiscStringArray *) **lines**

Returns the MiscStringArray object that the MiscShell is using to store individual lines of the process's output. You're on your honour not to muck around with it.

See also:

lineCount

- (int) **lineCount**

Returns the number of output lines received so far.

See also:

pause:

- **pause:***sender*

Sends **pause:** to the MiscSubprocess object.

See also:

read:

- **read:**(NXTypedStream *)*stream*

Your basic archiving method.

See also:

resume:

- **resume:***sender*

Sends **resume:** to the MiscSubprocess object.

See also:

runToCompletion

- (BOOL)**runToCompletion**

If true, the **executeScript:** method will launch the subprocess, read all of its output into the **lines** array, and wait for the script to execute before completing.   If false, the **executeScript:** method merely launches the subprocess and returns without waiting for it to complete.

See also:

script

- (MiscString *)**script**

Returns the shell script to be executed.

See also:

setAction:

- **setAction:**(SEL)*anAction*

Set the action message to be sent whenever the script produces a complete line of output.

See also:

setDelimiter:

- **setDelimiter:**(char)*c*

Sets the delimiter character to be used when splitting individual output lines into fields.

See also:

setExecArgs:::

- **setExecArgs:**(const char *) *a1:* (const char *)*a2 :*(const char *)*a3*

Sets the strings to be used by the custom subclass of MiscSubprocess when executing the child script. By default, these are "/bin/sh", "sh", "-c".

See also:

setRunToCompletion:

- **setRunToCompletion:**(BOOL)*c*

Sets the **runToCompletion** flag.

See also:

setScript:

- **setScript:**(MiscString *)*script*

Set the script to be executed to the indicated MiscString.

See also:

setTarget:

- **setTarget:***aTarget*

Set the target object to which messages should be sent after each line read.

See also:

stringValue

- (const char *)**stringValue**

Returns the last *complete* line of output read.

See also:

takeStdinFrom:

- **takeStdinFrom:***sender*

Asks *sender* for its stringValue, and writes that string to the standard input of the script. Presumably the script will then read this value.

See also:

target

- **target**

The target object to which target/action messages should be sent.

See also:

terminate:

- **terminate:***sender*

Sends the **terminate:** message to the MiscSubprocess.

See also:

write:

- **write:**(NXTypedStream *)*stream*

Your basic archiving method.

See also:

- writeToStdin:(const char *)*str*

Writes some data to the process's standard input.