Half-Pipe

A Java Command Line Shell

mik3hall@gmail.com

Half-Pipe

Half-Pipe is a command line shell written in java. It used to be called CommandLine and was part of what I used as my own IDE for a number of years. The other part was javacWrap, which was used for compiling java source code. I put time and use into both of these. They managed to meet my needs well enough. But with many other fine IDE's available I eventually discontinued supporting my own.

I am no shell expert so although Unix shells served as a model for this functionality I make no guarantees that it conforms to any actual standard way of doing any of this. I just thought it would be what most potential users would be comfortable with.

One idea I had concerning the CommandLine part was to use it for scripting. I had simple commands for running AppleScript using Runtime exec as well as a simple wrapper for on the fly compiling with javac and running java. I was aware though that there were things like BeanShell, JSR 223 in the offing, the Rhino java version of JavaScript and so on. I think jython, jacl was it?, and other packages go back to even earlier days now that I think about it. So, again, I started thinking about adding more scripting support into the CommandLine application. If you have a platform for skateboarding tricks, the more tricks you can do the better, no? So that is where a lot of the current effort should go and more on that should follow.

Legal to my understanding:

At least It in a basic sort of way HalfPipe should do most of the typical Unix shell type stuff. If you are familiar with that sort of thing then the following might give you some idea...

```
set java.* | grep version >temp
cat temp
System.in:3:java.runtime.version=1.7.0-ea-b211
System.in:4:java.version=1.7.0-ea
System.in:7:java.class.version=51.0
System.in:10:java.vm.version=21.0-b17
System.in:20:java.vm.specification.version=1.7
System.in:27:java.specification.version=1.7
exec ls -l temp
-rw-r--r--  1 mjh  staff  246 Oct 22 08:48 temp
rm temp
exec ls -l temp
ls: temp: No such file or directory
```

There is alias support and System properties serve somewhat as environment variables. See the command.properties file for most of the alias definitions.

```
set alias.who
alias.who=echo $user.name
who
mjh

set alias.versions "set java.* | grep version"
versions
System.in:3:java.runtime.version=1.7.0-ea-b211
System.in:4:java.version=1.7.0-ea
System.in:7:java.class.version=51.0
System.in:10:java.vm.version=21.0-b17
System.in:20:java.vm.specification.version=1.7
System.in:27:java.specification.version=1.7
```

More on alias

In doing some recent reading on scripting I came across something called currying. Roughly this seemed to be the idea of defining a function with some parameters and then wrapping another function around that one where you can pass in yet more parameters.
At the same time I have been looking into some of the serialization functionality that this application used to have to serialize it's own gui. I noticed that Eclipse is telling me that for many of my serializable classes I should of added the serialVersionUID.
So with this in mind I added the following to command.properties...

alias.serialver=exec serialver -classpath $java.class.path

The parsing support was already there but apparently untested so after a little debugging this finally worked...

```
serialver java.lang.String
java.lang.String:    static final long serialVersionUID = -6849794470754667710L;
```

To get back to the original point this could be considered currying? Or more simply providing some default parameters.

I googled Unix alias to see what functionality that provided. Thinking that pretty much that was just basically synonym. Something like, as a throwback to my own earlier rexx days, command.properties has...

```
alias.say=echo
```

But the alias wiki showed that it also supports providing default parameters. I have added something like that but currently only for '-' switch parameters.

So that if you didn't want to take the default application classpath the serialver alias provides you could instead go...

```
serialver -classpath /Volumes/mbvol/HalfPipe/halfpipe.jar utillib.serial.ParmedSerializableObject
utillib.serial.ParmedSerializableObject:    static final long serialVersionUID =
6437648825117615334L;
```

You could without too much difficulty write a command to Runtime exec serialver. I wrote classes specifically to facilitate doing this, one being org.cmdline.exec.RuntimeExecutable. As you can see I even gave these classes a package of their own. Customized versions were done, JavaRTExecutable and OsascriptRTExecutable, meant to make running those easier yet. A number of provided runtime exec'd commands are currently included with HalfPipe, open, shell, otool, etc.

If I'd done an alias exec before I might of skipped some of that. It allows you accomplish about the same thing without writing any java at all.

Support has now been added for positional parameters in alias definitions. command.properties includes a couple of examples. This simple test case...

```
alias.testparms=echo "${1} ${2} ${1}"
```

Should give...

```
testparms one two
one two one
```

A few, possibly more interesting, examples related to FScript and the OS X Scripting Bridge.

```
alias.app=fscript "${1} := SBApplication applicationWithBundleIdentifier:'${2}'"
alias.bundleid=as "get id of application "${1}""
alias.sysbrowse=fscript "sys browse${1}"
```

Ah, a new toy. Which can be used like...

```
bundleid itunes
com.apple.iTunes
app itunes com.apple.iTunes

fscript "itunes class"
ITunesApplication
fscript "itunes activate"
```

Actually, the sysbrowse alias doesn't completely work yet. I was trying to make the positional parameters optional but it doesn't work if it's omitted. It does work with the above by providing the parameter like so...

```
sysbrowse :itunes
```

Thats all fine for me, I can easily edit the command.properties file to mess with alias's. But how about you? I think the prior incarnation of the application had some HTML functionality that I have not so far continued in HalfPipe. Some of it involved native in a way no longer valid so I dropped it in this reincarnation. That appears to have included a HTML based properties editor. This may of involved self-modification of the command properties file directly in it's jar which I don't think now is a real good idea. It was one of those things probably fine for my own use but not the best for a distributed application.

However, I will at some point try to figure out exactly what I was doing and see if the HTML properties editor itself is worth the salvage work.

Meanwhile, to allow you to have your own alias's in a persistent way, beyond defining them in the shell for the current session only, I have added a 'load' command.

```
load filename
```

The alias functionality of HalfPipe is a little different compared to what other shells do but I'd like to think it almost provides a sort of shorthand scripting in and of itself.

Half-Pipe and Scripting

JSR 223 scripting

```
engines
ScriptEngineFactory Info
        Factory class:class org.jruby.embed.jsr223.JRubyEngineFactory
        Script Engine: JSR 223 JRuby Engine (1.7.4)
        Engine Alias: ruby
        Engine Alias: jruby
        Language: ruby (jruby 1.7.4)
ScriptEngineFactory Info
        Factory class:class com.sun.script.javascript.RhinoScriptEngineFactory
        Script Engine: Mozilla Rhino (1.7 release 3 PRERELEASE)
        Engine Alias: js
        Engine Alias: rhino
        Engine Alias: JavaScript
        Engine Alias: javascript
        Engine Alias: ECMAScript
        Engine Alias: ecmascript
        Language: ECMAScript (1.8)
ScriptEngineFactory Info
        Factory class:class us.hall.scripting.RhinoScriptEngineFactory
        Script Engine: Rhino JavaScript Script Engine (1.0.0)
        Engine Alias: javascript
        Engine Alias: js
        Engine Alias: mozrhino
        Language: JavaScript (VERSION_DEFAULT)
ScriptEngineFactory Info
        Factory class:class org.rosuda.jrs.RScriptFactory
        Script Engine: REngine (0.7)
        Engine Alias: REngine
        Language: R (2)
ScriptEngineFactory Info
        Factory class:class apple.applescript.AppleScriptEngineFactory
        Script Engine: AppleScriptEngine (1.1)
        Engine Alias: AppleScriptEngine
        Engine Alias: AppleScript
        Engine Alias: OSA
        Language: AppleScript (2.2.4)
```

The above shows the output of the Half-Pipe engines command. This command is actually just a few lines of code from the the JSR 223 documentation. Code snippets is part of what Half-Pipe is about. If you see a few lines of useful code you can wrap a simple command around them and always have them handy.

This shows the ways you might have JSR 223 scripting available. The AppleScriptEngine is the one currently available by default with the OS X openjdk implementation. If the Jruby OS X framework is installed you should automatically pick up the engine for that, which is probably the next easiest way to do JSR 223. For javascript I threw together a simple JSR 223 interface to mozilla Rhino, there is now also the openjdk install javascript. I have included the jars for an R interface engine as well.

Examples:

AppleScript:

I have done a little with standalone JSR 223 AppleScript previously outside of Half-Pipe located here StringTemplate and AppleScript

The JSR 223 version is now the *as* command.

```
as
set cd to (random number 100000) mod 52 + 1
return cd
13
```

For this one there was also something in place in the application previously using the native osascript command along with Runtime exec. So as an example of the old fashioned way of doing this...

```
org.cmdline.cmds.osascript
set cd to (random number 100000) mod 52 + 1
return cd
21
```

JRuby:

My understanding is some of the interest in the 1.7 mlvm effort was directed toward this. I am also trying to pick up a bit of the language. So I will be putting more time into this.

```
rbs
puts 42
puts 42.class
puts 42.class.class
42
Fixnum
Class
```

JavaScript / Rhino:

This uses my own JSR 223 interface classes to support this. Based mostly on this article Build your own scripting language for Java – JavaWorld and the Rhino Shell.java example including from there the 'print' function even though this is “not part of ECMA”. Not too bad getting something that sort of works anyhow. It should be noted that I'm a little familiar with browser based JavaScript, not at all Rhino.

```
js
var something = "else"
var num = 42
print(something,num);
else 42
```

R:

There is a JSR 223 interface to R. It is seeming a little off at the moment. This might be the new release of R I installed recently, 3.1.1, which is it seems the “Sock it to Me” version. Probably more to say on that when I've put more time into. Maybe just bring it more current but I think I had to make a couple modifications to get things working.

Other: java

dojava is a simple wrapper for a command similar to the above for doing java. It isn't JSR 223 based. It simply wraps a public static void main around the entered java, then compiles and runs it. It works the same in reading lines from the keyboard until CTRL-D is entered.

```
dojava
String greeting = "Hello,";
String comma = ", ";
String addressee = "World";
System.out.println(new StringBuilder(greeting).append(comma).append(addressee));
Hello,, World
dojava: done
```

Other: F-Script (OS X only)

A smalltalk like scripting interface to ObjectiveC Cocoa. Some functionality may not be there from a java shell. For example Core Image graphics to the java window didn't work but to a created window did. With the F-Script **object browser** we do know the java window is a AWTView. Try...

sys browse:[some Cocoa object] and click on “Select View” and then click on the java window.

F-Script is persistent, variables are saved somewhere and can be accessed later. See the descriptions for the fscript and jifs commands for more information.

```
fscript "NSDate date"
2011-11-26 15:00:37 -0600
```

More on Fscript

I updated my interface for this 08.23.14 to get it working again.

A couple things needed changing. I had changed the package for the FScript class but hadn't updated the JNI source to reflect this. I had to switch the code from using my classpath JNI loader – org.cmdline.common.NativeLoader. This code allows you to put a dynamic library anywhere in classpath and load it rather than requiring it to be set in LD_LIBRARY_PATH or java.library.path.

Instead now, it uses the usual Oracle java JNI loading by putting the dylib in the MacOS application directory and using 'load'. Otherwise you get a linkage error. I have run the code command line just as it should run from the application with...

```
java -cp
/Users/mjh/HalfPipe/HalfPipe7.app/Contents/Java/launcher.jar:/Users/mjh/HalfPipe/HalfPipe7.app/Content
s/JavaApp/halfpipe.jar:/Users/mjh/HalfPipe/HalfPipe7.app/Contents/Java/Classes us.hall.scripting.FScript
"NSDate date"
```

This is probably due to application classloading differences. This could be of interest if anyone actually wants to hack on the application. Otherwise, document the problem for my own future reference.

Some other F-Script related links that could be of interest...
FScript Home
Github download for 10.7, 10.8 and 10.9
OS X Frameworks

I install the FScript framework in /Library/Frameworks. Embedding in the application might be an option I haven't tried yet.

There is undoubtedly more that can be done with the JSR 223 interface. Hopefully, I will add functionality as I become more familiar with it. However, a couple scripting options already have more fully interactive interpreter functionality available.

Interactive scripting

The usual Ruby shell, to my still limited understanding, is irb, or with JRuby it's jirb. From some Googled JRuby demo code and another user example of adding the irb console to Eclipse I got this working with the Half-Pipe plumbing.

```
irb
irb(main):001:0>def greeting
puts 'Hello, World!'
end
nil
irb(main):004:0>greeting
Hello, World!
nil
irb(main):005:0>
```

From a java Swing text component it could look better than what it does, but what I currently have for styling command output isn't well suited to interactive commands.

The other scripting language that I know supports a interactive mode is Rhino. I partially based the JSR 223 interface on the Shell example. The following except for tweaking the packaging actually is the Shell example.

```
jijs
js> function greeting(g) {
print(g);
}
js> greeting("Hello, World!");
Hello, World!
js>
```

This is what I intend to do for interactive at this time. I need to use them myself to get a better understanding of the scripting languages and functionality.

Batch and mixed language scripting

Things should probably get a little more complex at this point. What the shell will provide for running larger and more complex scripts and getting back larger and more complex results when mixing script languages.

For this week I am simply adding a batch option to the js and rbs commands.

```
rbs /Volumes/mbvol/jruby-1.6.0.RC3/samples/thread.rb
abxyzc

js /Volumes/mbvol/rhino1_7R3/examples/enum.js
arg: /Volumes/mbvol/rhino1_7R3/examples/enum.js
0.0
1.0
2.0
```

JRuby, JavaScript and the HalfPipe shell

Added functionality to issue commandline invocations from within JavaScript and JRuby. Remember that part of the point of this exercise is as a learning experience for me. If my approach or implementations seem naïve and awkward, please bear with me, hopefully it will get better as I learn more.

For the shell we can of course just define a JavaScript function there. 'hp' is the function for HalfPipe.

```
jijs
js> hp('versions')
js> System.in:3:java.runtime.version=1.7.0-ea-b219
System.in:4:java.version=1.7.0-ea
System.in:7:java.class.version=51.0
System.in:10:java.vm.version=21.0-b17
System.in:20:java.vm.specification.version=1.7
System.in:27:java.specification.version=1.7
```

This is actually a variable args function.

```
hp('set','java.*','|','grep','version')
js> System.in:3:java.runtime.version=1.7.0-ea-b219
System.in:4:java.version=1.7.0-ea
System.in:7:java.class.version=51.0
System.in:10:java.vm.version=21.0-b17
System.in:20:java.vm.specification.version=1.7
System.in:27:java.specification.version=1.7
```

If building a command up from parts. Notice that the next prompt comes before the output is displayed. The shell execution is asynchronous. I started looking at this but the linkage is complex enough to make it difficult to come up with a quick synchronous invocation.

Since we use our own code for the JSR 223 interface it was possible to just define a builtin JavaScript function to the org.cmdline.scripting.RhinoScriptableObject class. Again, 'hp' for HalfPipe function.

```
js
hp('versions')
System.in:3:java.runtime.version=1.7.0-ea-b219
System.in:4:java.version=1.7.0-ea
System.in:7:java.class.version=51.0
System.in:10:java.vm.version=21.0-b17
System.in:20:java.vm.specification.version=1.7
System.in:27:java.specification.version=1.7
```

This ability to add a 'built-in' function to the JSR 223 seems nice enough to me that I may look at allowing you to subclass mine and plug in your subclass as the one to use for the JSR 223 interface.

For JRuby I added the functionality the same way for both the interactive shell and the JSR 223 interface.

```
rbs
HP.new.hp 'set ','java.* ','| ','grep ','version'
System.in:3:java.runtime.version=1.7.0-ea-b219
System.in:4:java.version=1.7.0-ea
System.in:7:java.class.version=51.0
System.in:10:java.vm.version=21.0-b17
System.in:20:java.vm.specification.version=1.7
System.in:27:java.specification.version=1.7
```

My implementation here may definitely not be the best. I use the same technique that I copied for the jirb interactive. Run a scriptlet to add code into the JRuby runtime. The code defines a 'HP' class with a 'hp' method. It uses splat for the variable args which needs a extra blank for a simple to_s conversion to make a proper command line string. It probably wouldn't be much more involved to append the blank in the method and I will probably do so soon. Not shown, but the invocations are again asynchronous.

It would of course be nice to have synchronous invocations if you want or maybe allow a callback for asynchronous results. I am also thinking using custom OutputStream replacements for System.out to allow printed results to be trapped that way.

So a sort of start of mixing things together, still considering rexx-like address command functionality as well.

Available Commands

alias	Manipulate shell alias's. Mostly I do this with the 'set' command these days. So this is pretty much obsolete and unused. Usage: alias *name value*
as	Reads AppleScript from the keyboard until CTRL-D, and then an OS X JSR 223 interface invokes the typed script.
cat	Print contents of named file or outputs from System.in if no filename is given. Usage: cat [*filename*]
classifier	Weka classifier information. Usage: classifier j48 trees
classifiers	Weka classifier lister. Usage: classifiers
dojava	Reads from keyboard until CTRL-D, wraps the entered text in a public static void main method, it then compiles and runs it.
engines	List available JSR 223 scripting engines. Based on Scripting for the Java Platform
env	Enumerates environment variables from System.getenv().
exec	Runtime exec interface. Usage: exec [*native command and arguments*]
exit	Command line shut down the application.
fb	Indicates likely presence of Flashback virus and shows related F-Secure URL.
fscript	OS X only JNI interface. Reads F-Script from the keyboard until CTRL-D, and then runs it using a JNI interface to the F-Script framework. Requires access to the F-Script framework, I installed at /Library/Frameworks. It also requires libfscript.jnilib, I am currently pointing to it with -Djava.library.path.
grep	Simple implementation of grep. Not sure anymore where I got it. Usage: grep *pattern [file]*
host	Default provides host name and ip address for localhost. Optionally prints the ip address(es) for a passed host name list. Usage: host [name1[name2 […]]]
jijs	Fully interactive Rhino JavaScript shell interface. Requires js.jar in classpath. See js for a currently less interactive JSR 223 Rhino scripting interface.
jirb	Fully interactive version of JRuby IRB interface. Requires JRuby jar in classpath and JRUBY_HOME set. See 'rbs' for a less interactive JSR 223 JRuby scripting interface.
js	Reads Javascript from the keyboard until CTRL-D, and then uses a Rhino JSR 223 interface to invoke the typed script.
kb	Print numeric and KeyEvent virtual key codes for entered keyboard characters. Current thought is this could be useful in allowing keyboard alias shortcuts. Not yet done.
load	Actual new command for HalfPipe. Load a external properties file into the application's System properties. Usage: load *filename*
osascript	Reads AppleScript from the keyboard until CTRL-D, and then Runtime exec invokes the native OS X command of the same name.
ps	Name meant to be suggestive of the Unix command. It actually is one going back to Java In A Nutshell to list thread information. Some special handling for 'Runner' threads. These are the thread wrappers for shell commands. Actually I think that might be all it shows now. I should probably add some options

	for other threads.
rbs	Reads Ruby from the the keyboard until CTRL-D, and then uses a JRuby JSR 223 interface to invoke the typed script. Usage: rbs [*filename* | *string to eval*]
seclist	List information on available crypto services.
serialver	Runtime exec the java serialver tool. See the **More on Alias** section.
set	'set' or display System property values. Allows wildcard pattern matching [*,?] for listing properties. Usage: set *[property [value] | property pattern]* No args displays all properties.
showc	Simple reflection display of a class file. I think an original Java In A Nutshell derived. Should be modernized with Java 5 annotations and the like sometime.
jnp2.SimpleWebBrowser	Usually used to display piped in html. This one is based on an example from Java Network Programming by Eliotte Rusty Harold. e.g. `echo "<h1><font color="red"> Hello, World!</font></h1>" | jnp2.SimpleWebBrowser`
uidefaults	List Swing properties.
whereis	Searches for the entered class. The 'smart' search used by the shell to locate executable commands without fully qualified package/class name. This used to be useful for JDK tools but appears less so these days... whereis jar whereis: com/sun/java/util/jar/pack/AdaptiveCoding.class found at /Library/Java/JavaVirtualMachines/JDK 1.7.0 Developer Preview.jdk/Contents/Home/jre/lib/rt.jar The command also currently puts out debugging println's because it seems broke when the class files are in directories as now seems usual for me using Eclipse.
xprotect	List the Safe Download version and last update time stamp.