Embedding FrAid

NOTICE: ""

Table of contents

""

1. Introduction

Qutie a few languages nowadays allow Java to be used in an interactive scripting environment. These include Groovy, BeanShell, Jython to name just a few. Here is how you can use FrAid from such an environment (the examples are tested with Groovy but they should run on any of the listed above languages or Java itself).

Note:

Some of the mentioned scripting environments use custom class loaders which have problem loading FrAid if the installation contains more than one jar file. If you get any such errors use the single jar FrAid installation.

2. Uses

Comment:	Plot a function:	Execute a function:	FrAid script required:

Execute it as a FrAid script	`import org.fraid.Scriptin`
	
	`s = new Scripting()`
	`s.exec("plot(sin);`
	Result:
		`print new org.fraid.Scriptin`
		Result: (0.9092974268256817 - 0.0i)
			Yes

Pass a subclass of ComplexFunction	`import org.fraid.graphics`
	`import org.fraid.complex.`
	
	`new plot().exec(sin)`
	Result:
		Result:	No

Pass a instance of a subclass of ComplexFunction	`import org.fraid.graphics`
	`import org.fraid.complex.`
	
	`new plot().exec(new sin())`
	Result:
		`import org.fraid.complex.`
		`print new sin().exec(2)`
		Result: (0.9092974268256817 - 0.0i)
			No

Get it from the symbol table	`import org.fraid.Scriptin import org.fraid.graphics import org.fraid.complex.*; s = new Scripting() new plot().exec(s.get_` Result:	`print org.fraid.Scriptin` Result: (0.9092974268256817 - 0.0i)	No	`(2)`
Define your own function	`import org.fraid.Scriptin import org.fraid.graphics import org.fraid.complex.*; s = new Scripting() my_f = s.def_f("f(x)=sin( new plot().exec(sin, my_f)` Result:	`print new org.fraid.Scriptin` Result: (1.0403617660940427 + 0.0i)	Yes	`(x)^cos(x);"`
Define and get your own function from the symbol table	`import org.fraid.Scriptin import org.fraid.graphics import org.fraid.complex.*; s = new Scripting() s.def_f("f(x)=sin( new plot().exec(sin, s.get_f("f"))` Result:	`s = new org.fraid.Scriptin s.def_f("f(x)=sin( print s.get_f("f").exec(` Result: (1.0403617660940427 + 0.0i)	Yes	
Define your own function within the scripting environment	`import org.fraid.function import`	`import org.fraid.function import`	No	

""

	`org.fraid.complex.import org.fraid.graphics.import org.netlib.math.co`
	`class MyFun extends ComplexFunction{ MyFun(){ super(1); } Complex invoke( Complex[] args ){ return args[0].sin().pow( } } new plot().exec( sin, new MyFun() )`
	Result:
		`org.fraid.complex.import org.netlib.math.co class MyFun extends ComplexFunction{ MyFun(){ super(1); } Complex invoke( Complex[] args ){ return args[0].sin().pow( } } print new MyFun().exec(2)`
		Result: (1.0403617660940427 + 0.0i)	

3. Other useful functions in org.fraid.Scripting and org.fraid.ScriptingGraphics

Description	Script	FrAid script required
Query the symbol table for all functions whith names starting wiht "si".	`print org.fraid.Scripting.quer` Result:[sinh:1, sin:1, sign:1]	No
Query the symbol table, select a function from the result and invoke it.	`s = new org.fraid.Scripting() q=s.query_f("si") for( k in q ){ if( k == "sin" ){ print s.get_f( k ). exec(2) } }` Result: (0.9092974268256817 - 0.0i)	No
Plot some graphics and then close all the windows.	`s = new org.fraid.Scripting() s.exec("plot(sin);\`	Yes

	`f(x,r)=r*x*(1-x);\`
	`orbit(f);")`
	`print`
	`org.fraid.ScriptingGraph`
	`org.fraid.ScriptingGraph`
	`)`
	Result: 2 //number of windows closed	

- Open a window;
- Plot something in it;
- Change the color, span and size of the window;
- Repaint.
	`import`
	`org.fraid.complex.functi`
	`import`
	`org.fraid.graphics.funct`
	`s = new`
	`org.fraid.Scripting()`
	`my_win =`
	`org.fraid.ScriptingGraph`
	`new plot().exec(`
	`my_win, sin )`
	`p = s.get_p(my_win)`
	`p.backgroundColor = new`
	`java.awt.Color(100,0,100`
	`p.upperLeft = new`
	`org.fraid.math.DoublePoi`
	`-10, 10 )`
	`p.bottomRight = new`
	`org.fraid.math.DoublePoi`
	`10, -10 )`
	`org.fraid.ScriptingGraph`
	`org.fraid.ScriptingGraph`
	Result:
		No
		)

- Open a FrAid Console and work with the interpreter directly.	`prog = "plot(sin);"`
	`org.fraid.ScriptingConso`
	`prog )  //prog can be`
	`empty ("")`
	Result:
		Yes

""

""