

Ruby Functions from the SAS® DataStep

Daniel Olguin, First Coast Service Options, Jacksonville, FL

ABSTRACT

Are you tired of waiting on SAS® to provide user-defined functions to the DataStep? Incensed at the limitations that will still exist for user-defined functions when SAS 9.2 is finally released? With 3 relatively easy and painless steps you can add Ruby, a programmer friendly, Object-Oriented language, to your SAS Tool-Box. This paper will examine examples of using Ruby from the DataStep.

Keywords: Ruby, JRuby, open-source, DataStep, user-defined, function

INTRODUCTION

JRUBY

JRuby is a port of the Ruby language to the Java JVM. Ruby is an OO language with functional programming capabilities. Its designer, Yukihiro Matsumoto, known as Matz to the Ruby community, created a language with a clean, elegant design and minimal syntax for clarity in the hopes of providing programmers with a language that is a joy to use. JRuby version 1.0 was released in June of 2007 and is now at 1.1.2. It is compatible with Ruby 1.8.5.

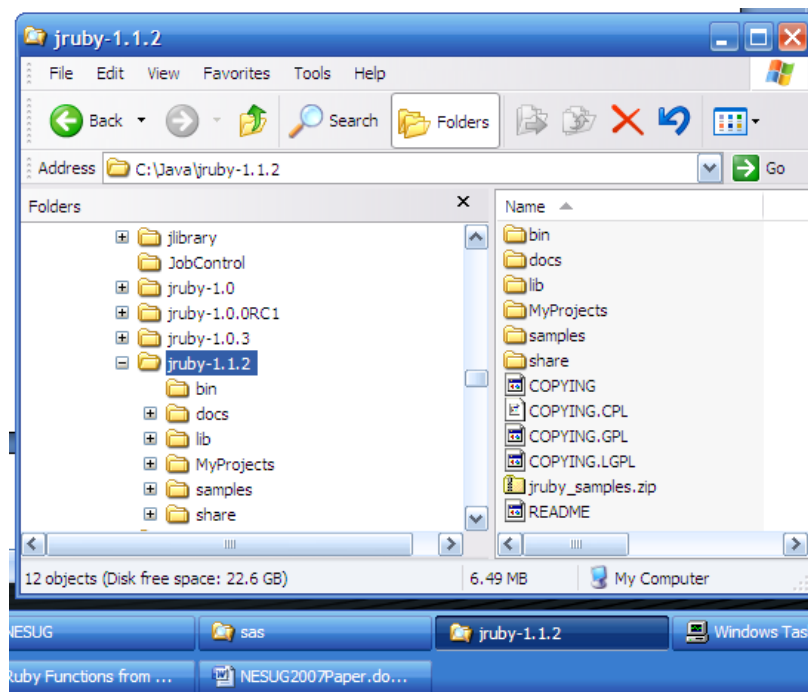
Adding JRuby (and therefore Ruby) to your SAS DataStep is done in 3 steps:

- 1) Download JRuby and Install (Requires Java installed. Tested with both 1.4 and 1.5 but 1.5 is preferable.)
- 2) Modify the SAS configuration file
- 3) Create an Adapter to handle the communication between SAS and JRuby

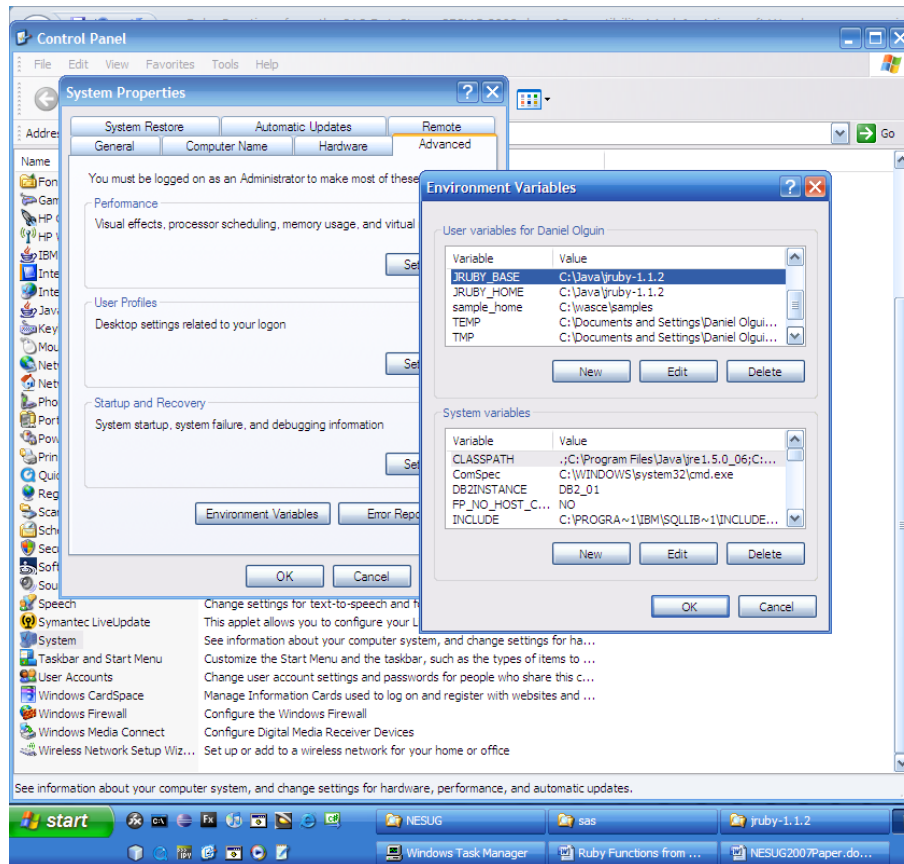
DOWNLOAD AND INSTALL JRUBY

The JRuby code is located at: <http://dist.codehaus.org/jruby/>

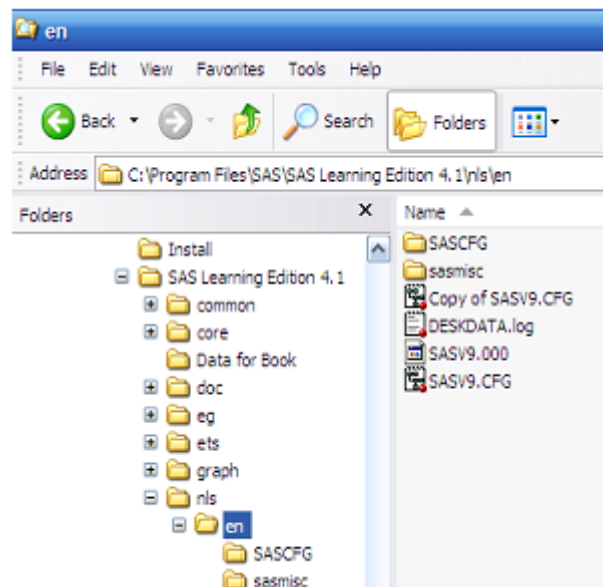
Download and unzip.



Set the Environment Variables for JRuby Home.



Locate the SASV9.cfg



Important: Make a backup!

Modify the -JREOPTIONS

SASV9.cfg (partial listing)

```

...

-JREOPTIONS=(-Djava.class.path="C:\Java\jruby-1.1.2\lib\myjruby.jar;C:\Java\jruby-
1.1.2\lib\jruby-complete-1.1.2.jar"
            -Dsas.jre=public
            )

...
```

The java.class.path needs to include at least 2 jars, the jar containing the adapter (in my case myjruby.jar), and the jruby-complete jar.

If you use a Java version other than the SAS private jre then set sas.jre=public. Lastly make sure you remove the reference to the SAS classloader.

The adapter or mediator can be quite simple. It's first task is to get an instance of the Ruby runtime. The rest can be as simple as having the runtime eval the script and then casting to the appropriate return value.

SASRubyMediator.java: (partial listing)

```

public class SASRubyMediator {
    private Ruby runtime;
    public String Init(){
...
        runtime = Ruby.newInstance();
...
    }
    public Long evalToLong(String script){
        final IRubyObject objRuby = runtime.evalScriptlet(script);
        return (Long) JavaEmbedUtils.rubyToJava(runtime, objRuby,
Long.class);
    }
    public Integer evalToInteger(String script){
        final IRubyObject objRuby = runtime.evalScriptlet(script);
        return (Integer) JavaEmbedUtils.rubyToJava(runtime, objRuby,
Integer.class);
    }
...
}
```

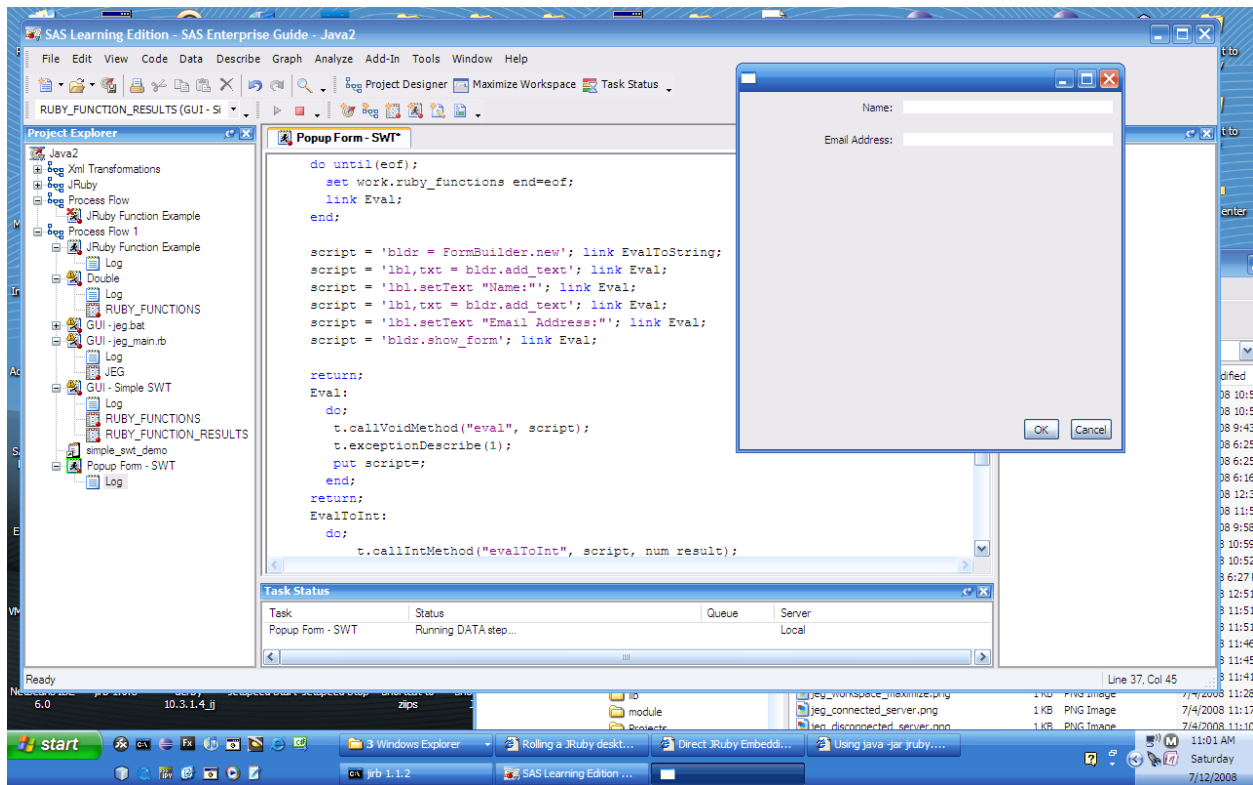
The mediator is initialized at `_N_ = 1`. Then functions can be defined and used (i.e. evaluated).

```

data work.ruby_functions;
  length script $2000 x $10;
  if _n_ = 1 then
    do;
      declare JavaObj t ('SASRubyMediator');
      t.exceptionDescribe(1);
      t.callStringMethod("Init");
      t.exceptionDescribe(1);
    end;
  retain t;
  length result 8 myscript $100;
  script = "def double(x) " ||
           "  x * 2 " ||
           "end; ";
  link DefineFunction;
  script = "double(15); "; link EvalToInt;
  x = strip(result);
  put x=;
  return;
DefineFunction:
  do;
    t.callVoidMethod("eval", script);
    t.exceptionDescribe(1);
  end;
  return;
EvalToInt:
  do;
    t.callIntMethod("evalToInt", script, result);
    t.exceptionDescribe(1);
  end;
  return;
run;

```

Since the scripts are simple text, they can be written inline, read in as datalines to the DataStep, or read as a file.



A form dynamically generated from the DataStep is shown above.

This demonstrates just a small part of the potential that JRuby provides. With its access to Java, JRuby makes over a decades worth of open-source projects available to the DataStep programmer. Well known and accessible open-source projects include:

- 1) Scheduling – Quartz
- 2) GUIs - Swing, SWT, Java2D
- 3) DataMining – Weka
- 4) Visualization – Processing
- 5) Rules Engine – JBoss Rules (aka Drools)

This is just a small sample of the free, open-source projects available. Having the right tools makes anything possible!

CONCLUSION

Thanks to JRuby and the javaobj, SAS DataStep programmers can write and execute their own functions, define and use OO classes, create GUIs, in short, do anything programmatically imaginable! Since JRuby runs on the JVM you have access to any and all of the open-source Java projects that exist, but with much less pain than if you had to work with Java directly all the time.

REFERENCES

Greetings from the Edge: Using javaobj in DATA Step by Richard A. DeVenezia, Remsen, NY

<http://www2.sas.com/proceedings/sugi29/033-29.pdf>

A Conversation with Yukihiro Matsumoto, <http://www.artima.com/intv/ruby.html>

Quartz <http://www.opensymphony.com/quartz/>

Weka http://weka.sourceforge.net/wiki/index.php/Use_Weka_in_your_Java_code

Processing <http://processing.org/>

Rules (aka Drools) <http://www.jboss.com/downloads/index>

RECOMMENDED READING

The Ruby Programming Language, David Flanagan & Yukihiro Matsumoto, O'Reilly Media, Inc. 2008

Ruby Cookbook, Lucas Carlson & Leonard Richardson, O'Reilly Media, Inc. 2006

Design Patterns in Ruby, Russ Olsen, Addison-Wesley Inc., 2008

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Daniel Olguin
2408 Spring Vale RD
Jacksonville, FL 32246
E-mail: olguind@bellsouth.net

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

Complete Program Listings may be downloaded from <http://olguind.clearwire.net/downloads>