

Simple SAS® Web Services with Ruby

Daniel Olguin, First Coast Service Options, Jacksonville, FL

ABSTRACT

You have an investment in SAS® programs and data and you'd like to make them available over the World Wide Web. If you're a Java programmer, SAS has examples for you. However, if you're not a Java programmer do you then have to climb Mt. Java and endure the Java police along the way just to make your SAS resources available on the Web? Luckily for you the answer is no! This paper will present a simple method for producing RESTful Web services for SAS through JRuby.

Keywords: Ruby, JRuby, Restful, web service, open-source

INTRODUCTION

In June 2007 Sun released JRuby 1.0, a port of the Ruby language to the Java Virtual Machine (JVM). Ruby is a powerful, programmer friendly, object-oriented language. Being able to run on the JVM means it's available everywhere Java is. It also means that it can take advantage of the broad Java ecosystem. Its ease of use makes quick work of what would otherwise be tedious coding in Java. Since SAS is a strong supporter of the use of Java there are many ways to interface SAS and Ruby.

SETUP

To run the Web Services described in this paper you'll need the following:

- 1) JRuby (Any version since 1.0. Current version at the time of this writing is 1.1.2)
 - a. Installation is easy. See my SESUG 2008 paper, 'Ruby Functions from the SAS® DataStep'.
- 2) Jetty an open-source Java based servlet container.
 - a. Installation is a simple as downloading from the Jetty download page and unzipping.
- 3) Something to communicate with SAS. SAS Integration Technologies® provided jars were used to test.

The layout for the Web Services provider is a src folder with a lib folder for the jars a module folder for ruby code and any optional folders for expanded functionality.

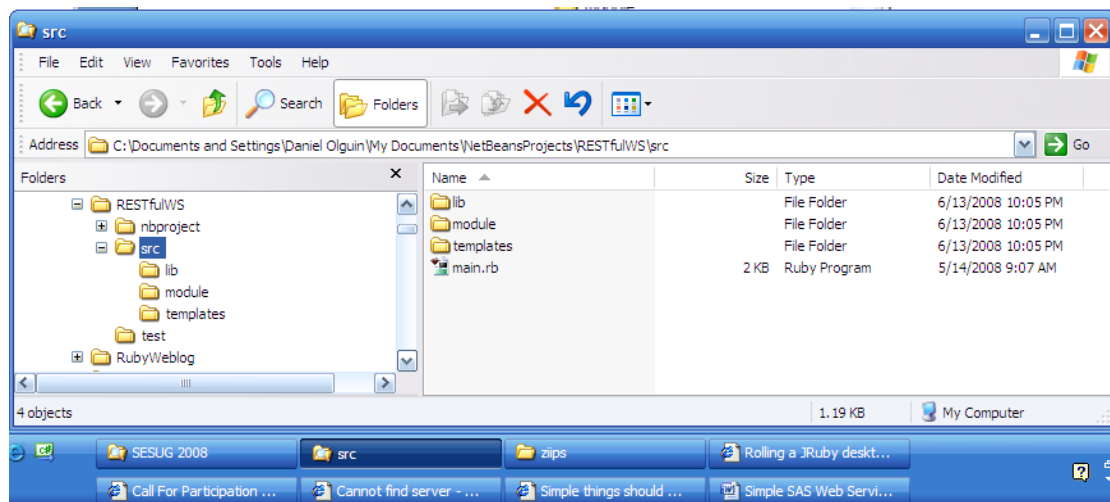


Figure 1. Folder Layout for Web Services provider

Copy the jars from the JRuby and Jetty installation directories to the lib folder for the project. If you're using SAS supplied jars, copy them there as well.

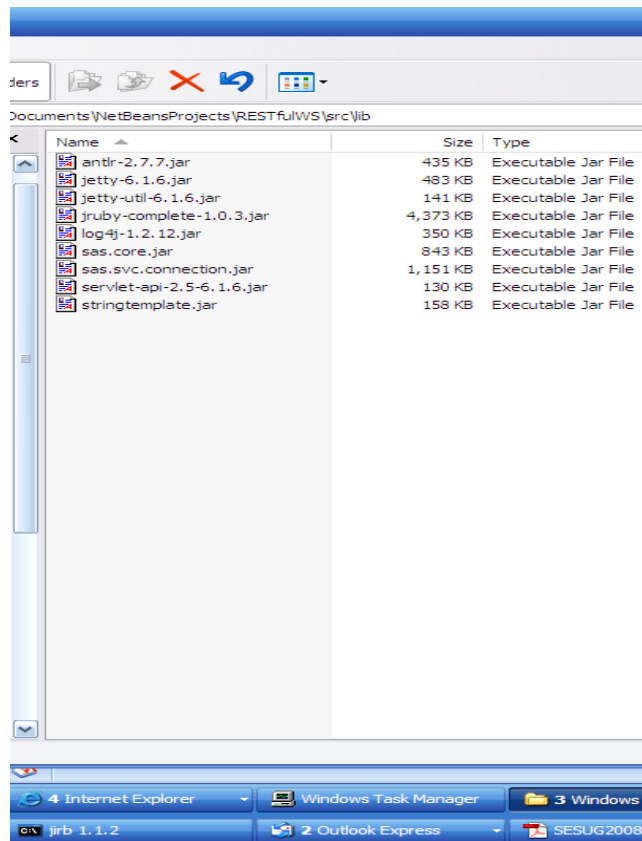


Figure 2. Example lib folders containing jar files

The antlr-2.7.7.jar and the stringtemplate.jar are from Terence Parr's Antlr and StringTemplate open-source projects and are individually useful but not required for this project. The jetty jars are two of the jars that are necessary and provide the servlet container functions. The jruby-complete jar provides the access to the JRuby language. The log4j jar is from the Apache Jakarta's Log4J project to provide logging for Java applications. The sas jars are from SAS Integration Technologies and provide access to SAS. Lastly, the servlet-api jar provides the Java classes necessary for creating servlets.

Jetty Server

The file `main.rb` in the `src` folder contains the core of the server code. It is less than 50 lines; two thirds of which is actual code. A server is instantiated listening on port 8080 and two contexts are created; one for retrieval of static content and one for the Web Service provider, `SASServlet`.

A `DRb` server is also started in the example in case the client is Ruby. If so, `DRb` offers a separate and alternative means of distributed communication.

```
main.rb (partial listing)

class SimpleServer
  attr_accessor :server
  def initialize(host)
    @server = Jetty::Server.new 8080
    contexts = Jetty::ContextHandlerCollection.new

    context = Jetty::Context.new(contexts, "/", Jetty::Context::SESSIONS)
    context.addServlet("org.mortbay.jetty.servlet.DefaultServlet", "/");
    context.setResourceBase ".."

    context =
Jetty::Context.new(contexts, "/sas", Jetty::Context::SESSIONS);
    context.addServlet(Jetty::ServletHolder.new(SASServlet.new), "/*");

    @server.setHandler(contexts)

    puts "Starting Jetty..."
    @server.start
    puts "Ready!"

    spawner = ObjectSpawner.new "#{host}"
    puts "Starting ObjectSpawner..."
    DRb.start_service("druby://#{host}:9000", spawner)
    puts "Ready!"
    DRb.thread.join

  end
end
```

SASServlet

The SASServlet is a subclass of the standard Java HttpServlet but Ruby style. Both doGet and doPost simply hand-off to the routing routine identify_request which uses regular expressions to parse the URL and decide on a course of action. Moving the routing to a dispatch table is an obvious alternative.

```
sas_servlet.rb (partial listing)

class SASServlet < ServletAPI::HttpServlet
  def doGet(request, response)
    ...
    identify_request(request, response)
  end
  def doPost(request, response)
    ...
    identify_request(request, response)
  end
  def identify_request(request, response)
    path = request.getPathInfo
    # Expecting a SAS Server (host) name and possibly more
    sas_server,remaining = /^\/([A-Za-z0-9_\.]*)\/?(.*)$/ .match(path) [1..2]
    puts "SAS Server: #{sas_server}; Remaining: #{remaining}"
    if !validServer(sas_server)
      malformed_request(response,"Unknown Server")
      request.setHandled(true)
      return
    end
    case
    when /^(login|logon|connect)$/ .match(remaining)
      then sas_connect(request, response, sas_server)
    when /^(logoff|disconnect)$/ .match(remaining)
      then sas_disconnect(request, response, sas_server)
    when /^(execute|run)$/ .match(remaining)
      then sas_execute(request, response, sas_server)
    when /^(log)$/ .match(remaining)
      then sas_log(request, response, sas_server)
    ...
  end
end
```

A simple .bat file on Windows or .sh file on Unix launches the server.

```
main.bat (partial listing)

c:\Java\jruby-1.1.2\bin\jruby main.rb 1>main.log 2>main.err.log

pause ...
```

Assuming your Server is listening on port 8080 on localhost, the service can be accessed from the following URL:

<http://localhost:8080/sas/myserver/connect>

where **myserver** is the name of the server where you have SAS installed.

CONCLUSION

Open-source software provides us the tools to create the applications we need with the features we want on the schedule of our choosing. A strong, thriving open-source community would benefit both SAS and its customers. Open-Source Jetty has provided us with a simple to use, embeddable, servlet container. Open-source JRuby has provided us with an elegant programming language for developing our application with a minimal amount of busy-work.

REFERENCES

Jetty Server homepage <http://www.mortbay.org/jetty-6/>

JRuby homepage <http://jruby.codehaus.org/>

Netbeans homepage <http://www.netbeans.org/> (An excellent open-source IDE supporting JRuby development.)

SAS support Website

http://support.sas.com/rnd/itech/doc9/dev_guide/dist-obj/javacInt/javaprogram/workfact/workfact_direct.html

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>