

Paper 493-2013

Writing a Useful Groovy Program When All You Know about Groovy Is How to Spell It

Jack Hamilton, Division of Research, Kaiser Permanente, Oakland, California

ABSTRACT

SAS® is a powerful programming system, but it can't do everything. Sometimes you have to go beyond what SAS provides. There are several built-in mechanisms for doing this, and one of the newest is PROC GROOVY. It sounds like a product of San Francisco's Haight-Ashbury, but it's actually a programming language based on another product with San Francisco Bay area roots, Java. You can think of it as a simplified, easier to use version of Java -- simplified enough that you can put together a useful PROC GROOVY program from Internet examples without knowing anything about the language. This presentation focuses on handling directories and ZIP files, but many other things are possible.

INTRODUCTION

One of the items that jumps out of the New Features documentation, because of its quirky name, is PROC GROOVY. Maybe it does something interesting. But what?

The first place to look is the online documentation, at the even more quirky URL

<http://support.sas.com/documentation/cdl/en/proc/65145/HTML/default/viewer.htm#p1x8agymI9gten1ocziihptcjz.htm>

What does it say?

Groovy is a dynamic language that runs on the Java Virtual Machine (JVM). PROC GROOVY enables SAS code to execute Groovy code on the JVM.

PROC GROOVY can run Groovy statements that are written as part of your SAS code, and it can run statements that are in files that you specify with PROC GROOVY commands. It can parse Groovy statements into Groovy Class objects, and run these objects or make them available to other PROC GROOVY statements or Java DATA Step Objects. You can also use PROC GROOVY to update your CLASSPATH environment variable with additional CLASSPATH strings or filerefs to jar files.

OK, it's some kind of language that runs within SAS, and it's related to Java in some way. Java can do lots of interesting stuff in addition to its constant string of security holes. So maybe this is worth pursuing.

Off to the interwebs. The first place to look is Wikipedia:

Groovy is an object-oriented programming language for the Java platform. It is a dynamic language with features similar to those of Python, Ruby, Perl, and Smalltalk. It can be used as a scripting language for the Java Platform, is dynamically compiled to Java Virtual Machine (JVM) bytecode, and interoperates with other Java code and libraries. Groovy uses a Java-like bracket syntax. Most Java code is also syntactically valid Groovy.

That sounds promising. It's a scripting language, and it's similar to a couple of other scripting languages. Scripting languages can usually do interesting things with the operating system, such as crawling through directories and zipping files, and I'm always looking for better ways to do those things because, frankly, what SAS has built in isn't very good.

GIVING IT A TRY

I don't know anything about Groovy, and very little about Java, so let's try the example in the SAS manual:

```
proc groovy classpath=cp;
  submit;
  class Speaker {
    def Speaker() {
      println "----> ctor"
    }
    def main( args ) {
      println "----> main"
    }
  }
  endsubmit;
quit;
```

Here's what we get (SAS 9.3 TS1M0 on Windows 7):

```
1   proc
1   !   groovy classpath=cp;
ERROR: Logical name is not available.
ERROR: Failed to get the physical path from fileref CP.
2   submit;
3   class Speaker {
4   def Speaker() {
5   println "----> ctor"
6   }
7   def main( args ) {
8   println "----> main"
9   }
10  }
11  endsubmit;
----> ctor
----> main
NOTE: The SUBMIT command completed.
12  quit;

NOTE: The SAS System stopped processing this step because of errors.
```

This is not an encouraging start.

From here on, I will omit the "proc groovy" and "quit" lines from most log listings.

BACK TO THE WEB - READING A ZIP FILE

But there must be some working examples of Groovy code out there. I know how to write ZIP files in SAS, but not how to read them, so that's what I'll look for. I Googled "groovy read zip file".

The first result in that search was at the very useful site StackOverflow

<http://stackoverflow.com/questions/645847/unzip-archive-with-groovy>

It had a simple 4 line example:

```
def zipFile = new java.util.zip.ZipFile(new File('some.zip'))

zipFile.entries().each {
  println zipFile.getInputStream(it).text
}
```

As it happens, I had a small ZIP file at the ready, so I replaced 'some.zip' with 'W:\WUSS2012\WUSS2012.zip', put the code inside PROC GROOVY using the syntax in the earlier example, took out the CLASSPATH option, and ran it:

```

23     submit;
24     def zipFile = new java.util.zip.ZipFile(new
File('W:\WUSS2012\WUSS2012.zip'))
25
26     zipFile.entries().each {
27         println zipFile.getInputStream(it).text
28     }
29     endsubmit;
ERROR: The SUBMIT command failed.
org.codehaus.groovy.control.MultipleCompilationErrorsException: startup failed:
Script3.groovy: 1: unexpected char: '\' @ line 1, column 57.
    .util.zip.ZipFile(new File('W:\WUSS2012\

```

OK, not so good, but at least there's a reasonable error message - something to do with the backslash. I know that Java is kind of Unix based, and Unix likes forward slashes in file names, so let's try that:

```

257     submit;
258     def zipFile = new java.util.zip.ZipFile(new
File('W:/WUSS2012/WUSS2012.zip'))
259     zipFile.entries().each {
260         println zipFile.getInputStream(it).text
261     }
262     endsubmit;

```

```

%PDF-1.5
%µµµµ
1 0 obj
<</Type/Catalog/Pages 2 0 R/Lang(en-US) /StructTreeRoot 167 0
R/MarkInfo<</Marked true>>>>
endobj
2 0 obj
<</Type/Pages/Count 13/Kids[ 3 0 R 29 0 R 30 0 R 31 0 R 35 0 R 42 0 R 43 0 R 44
0 R 45 0 R 46 0
R 51 0 R 53 0 R 57 0 R ] >>
[... lines omitted ]

```

This time, there's a lot of extra *stuff* in the log. I recognize it as the beginning of a PDF file, and I know there's a PDF file in that ZIP file. Success!

Well, of a sort. I want a list of the names, and if I wanted the contents I probably wouldn't want them to just print in the log, but at least it's a proof of concept.

At this point, I decided to look around for some consolidated documentation, something similar to the Perl and Python cookbooks, so I wouldn't have to learn more than the bare essentials. Looking around on the web, I found a small e-book company, The Pragmatic Bookshelf, that publishes a number of such books, including one on Groovy called

Groovy Recipes: Greasing the Wheels of Java, <<http://pragprog.com/book/sdgrvr/groovy-recipes>>.

The description says "Each recipe in Groovy Recipes begins with a concise code example for a quick start, followed by in-depth explanation in plain English." The example looked like it might be understood by ordinary mortals, so I bought the book and downloaded it in Mobi format (it's also available in epub and PDF, and can automatically load a book to Dropbox, Kindle, or Readmill if you wish).

The book starts with a long explanation of the design of Groovy, how it's different from Java, and how to use it in various environments, but I skipped most of that and went to Chapter 6, "File Tricks". The first example was a two-liner that does a directory listing. I changed the directory name and stuck it into SAS:

```

286     submit;
287         new File("W:/WUSS2012").eachFile{file ->
288             println file
289         }
290     endsubmit;
W:\WUSS2012\7zipresult.png
W:\WUSS2012\CGF_55.pdf
W:\WUSS2012\CGF_57.pdf
[... lines omitted ]
NOTE: The SUBMIT command completed.

```

Ta Da! It works!

Looking at the difference between this and the previous example that printed contents, I noticed that the println in the previous example included something not in the second example, ".getInputStream(it)". So what happens if we take that out?

```

327     submit;
328         def zipFile = new java.util.zip.ZipFile(new
File('W:/WUSS2012/WUSS2012.zip'))
329         zipFile.entries().each {
330             println zipFile
331         }
332     endsubmit;
java.util.zip.ZipFile@5e9db7
java.util.zip.ZipFile@5e9db7
java.util.zip.ZipFile@5e9db7
java.util.zip.ZipFile@5e9db7
java.util.zip.ZipFile@5e9db7
java.util.zip.ZipFile@5e9db7
java.util.zip.ZipFile@5e9db7
NOTE: The SUBMIT command completed.

```

There are seven lines of output, and there are seven files in the ZIP file, so we're on the right track. Now we just need to print the file name instead of that mysterious code.

I look through more examples, not trying to understand them but just get a feel for the syntax, and I see a few examples that use "file.name" instead of just "file". So let's try that:

```

337     submit;
338         def zipFile = new java.util.zip.ZipFile(new
File('W:/WUSS2012/WUSS2012.zip'))
339         zipFile.entries().each {
340             println zipFile.name
341         }
342     endsubmit;
W:\WUSS2012\WUSS2012.zip
W:\WUSS2012\WUSS2012.zip
W:\WUSS2012\WUSS2012.zip
W:\WUSS2012\WUSS2012.zip
W:\WUSS2012\WUSS2012.zip
W:\WUSS2012\WUSS2012.zip
W:\WUSS2012\WUSS2012.zip
NOTE: The SUBMIT command completed.

```

OK, we're not there yet, but at least the guess that I could add ".name" to the end was correct.

Maybe we need outside help here. Let's go back to Mr. Google and search for "groovy zipFile.entries".

The first three results don't look promising, but the fourth is:

[Search in zipfile : Groovy Almanac
groovy-almanac.org/search-in-zipfile/](http://groovy-almanac.org/search-in-zipfile/)

```
Search. def searchstr = " 1 .txt" def zipfile = new ZipFile("test.zip") zipfile.entries().each{ entry-> if
(entry.name =~ searchstr){ println "$entry.name" } }. view plain ...
```

That sounds good. The code has a mix of familiar and unfamiliar stuff:

```
def searchstr = "1.txt"
    def zipfile = new ZipFile("test.zip")
    zipfile.entries().each{ entry->
        if (entry.name =~ searchstr){
            println "$entry.name"
        }
    }
```

Let's change the zipfile definition to match what we used earlier and try it:

```
587     submit;
588         def searchstr = "1.txt"
589         def zipfile = new java.util.zip.ZipFile(new
File('W:/WUSS2012/WUSS2012.zip'))
590         zipfile.entries().each{ entry->
591             if (entry.name =~ searchstr){
592                 println "$entry.name"
593             }
594         }
595     endsubmit;
NOTE: The SUBMIT command completed.
```

No output, but there weren't any matching files to print. More important is the lack of error messages. Let's try again with the search string and if statement removed:

```
600     submit;
601         def zipfile = new java.util.zip.ZipFile(new
File('W:/WUSS2012/WUSS2012.zip'))
602         zipfile.entries().each{ entry->
603             println "$entry.name"
604         }
605     endsubmit;
wuss2012/._WritersGuidelines2012.pdf
wuss2012/._WUSS2012_PresentersCopyrightForm.pdf
wuss2012/._WUSS2012_PresentersFAQs.pdf
wuss2012/WritersGuidelines2012.pdf
wuss2012/WUSS2012_PaperTemplate.doc
wuss2012/WUSS2012_PresentersCopyrightForm.pdf
wuss2012/WUSS2012_PresentersFAQs.pdf
NOTE: The SUBMIT command completed.
```

Aha! Now we have what we want. Now all we have to do is get it somewhere SAS can use it, which will come later in the paper.

An important thing to note here is that I don't know what a lot of this code is doing, and I don't have to. All I need to know is that it works. I can guess that the ".each" does some kind of iteration, and "entry->" is probably responsible for setting the value of "\$entry.name", but I don't know the exact syntax, and I don't know what I could use besides ".name". I could look it up, but I don't need to. That would be learning more about Java and Groovy than I need to.

Note that the title of this paper is *Writing a Useful Groovy Program When All You Know about Groovy Is How to Spell It*. It's not *Proc Groovy for Dummies* or *Proc Groovy for the Timid*. You can start using Proc Groovy without knowing anything about it, but you have to be willing to do some web searching, and draw analogies from relevant parts of SAS, and be willing to keep going even when the first 2 or 3 or 10 things you try don't work.

And also, even that earlier attempt that ended up printing the contents of the file into the log may turn out to be useful later, when you actually want the contents instead of the names.

DOCUMENTATION

At some point, you will probably need to look at real technical documentation just to learn keywords, even if you don't have to learn Groovy syntax. The main site for Groovy is <http://groovy.codehaus.org/>, and for documentation, <http://groovy.codehaus.org/Documentation>.

Many Groovy programs explicitly use Java functions, as in the ZIP examples above. You can see one of many Java documentation pages at <http://docs.oracle.com/javase/6/docs/api/>.

For example, to see what else you can do with a zip file, go to that Java page:

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im	Provides classes and interfaces for the input method framework.
java.awt.im.spi	Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.
java.awt.image	Provides classes for creating and modifying images.
java.awt.image.renderable	Provides classes and interfaces for producing rendering-independent images.

The section you want to go to is "All Classes" at the bottom left. Scroll down to `ZipEntry` and click on it. The right side will change to a description of the `ZipEntry` class:

Class ZipEntry

`java.lang.Object`
↳ `java.util.zip.ZipEntry`

All Implemented Interfaces:
[Cloneable](#)

Direct Known Subclasses:
[JarEntry](#)

```
public class ZipEntry
  extends Object
  implements Cloneable
```

This class is used to represent a ZIP file entry.

Field Summary

Field	Summary
<code>STATIC INT</code>	<code>CENATT</code>

Click on "Method" at the top left to get a list of methods. As in SAS, methods in Groovy and Java act on objects, such as a ZIP file object.

Two of the methods near the top are `getName()` and `getSize()`. Let's try those. They require slightly different syntax than the earlier example, but this syntax is actually more similar to what SAS uses for objects than the earlier example was

```

656     submit;
657         def zipfile = new java.util.zip.ZipFile(new
File('W:/WUSS2012/WUSS2012.zip'))
658         zipfile.entries().each{ entry->
659             println entry.getName() + ' ' + entry.getSize()
660         }
661     endsubmit;
wuss2012/._WritersGuidelines2012.pdf 4096
wuss2012/._WUSS2012_PresentersCopyrightForm.pdf 4096
wuss2012/._WUSS2012_PresentersFAQs.pdf 4096
wuss2012/WritersGuidelines2012.pdf 764690
wuss2012/WUSS2012_PaperTemplate.doc 167936
wuss2012/WUSS2012_PresentersCopyrightForm.pdf 158222
wuss2012/WUSS2012_PresentersFAQs.pdf 199185
NOTE: The SUBMIT command completed.

```

It's not a pretty format, but now you can get the name and the size.

READING A DIRECTORY STRUCTURE

Reading a directory structure is something I need to do fairly often. I have a SAS-based method for doing it (see http://www.sascommunity.org/wiki/Obtaining_A_List_of_Files_In_A_Directory_Using_SAS_Functions), but it's always good to have options.

First, let's read the files in a single directory. Searching for "groovy read file directory" finds this page with a nice two line function, <http://groovy-almanac.org/list-directory-contents/>, and it's easy to get a Proc Groovy program out of that:

```

694     submit;
695         new File('W:/WUSS2012/').eachFile() { file->
696             println file.getName()
697         }
698     endsubmit;
7zipresult.png
CGF_55.pdf
CGF_57.pdf
Corrected-Filenames-Recurse.sas
Corrected-Filenames.sas
filenames.sas
[... lines omitted ]
NOTE: The SUBMIT command completed.

```

That was easy.

And that web site, groovy-almanac.org, has lots of examples. At the bottom of screen where I found the code above, there's a link to a way to do another good file-ish thing, a recursive file listing, < <http://groovy-almanac.org/list-all-files-recursively/>>:

```

714     submit;
715         new File('W:/WUSS2012/recurse').eachFileRecurse() { file ->
716             println file.getPath()
717         }
718     endsubmit;
W:\WUSS2012\recurse\file1.txt
W:\WUSS2012\recurse\recurse2
W:\WUSS2012\recurse\recurse2\file2.txt
W:\WUSS2012\recurse\recurse2\recurse3
W:\WUSS2012\recurse\recurse2\recurse3\file3.txt
NOTE: The SUBMIT command completed.

```

Another example uses something called AntBuilder. It has a different syntax, and it lets you easily filter your results. As in SAS, it seems there is always more than one way to do a given task..< <http://groovy-almanac.org/list-directory-contents-with-antbuilder/>>.

```

796     submit;
797         def ant = new AntBuilder()
798         def list = ant.fileScanner {
799             fileset(dir:"W:/WUSS2012")
800         }
801         println "First listing"
802         list.each() { file ->
803             println "    sizeof ${file.getName()} is ${file.length()}
bytes"
804         }
805         list = ant.fileScanner {
806             fileset(dir:"W:/WUSS2012") {
807                 include(name:"*.sas")
808                 exclude(name:"*.pdf")
809             }
810         }
811         println "Second listing"
812         list.each { file ->
813             println "    sizeof ${file.name} is ${file.length()} bytes"
814         }
815     endsubmit;
First listing
    sizeof 7zipresult.png is 45564 bytes
    sizeof CGF_55.pdf is 273610 bytes
[... lines omitted ]
Second listing
    sizeof Corrected-FileNames-Recurse.sas is 4420 bytes
    sizeof Corrected-FileNames.sas is 2041 bytes
[... lines omitted ]
NOTE: The SUBMIT command completed.

```

You can see that the second listing includes only .sas files, as requested.

ASKING FOR HELP

The examples above have all been run under Windows. But when I ran the last example under Solaris, I saw this:

```

java.lang.NoClassDefFoundError: org/apache/tools/ant/BuildException
    at java.lang.Class.forName0(Native Method)
    at java.lang.Class.forName(Unknown Source)
[... lines omitted ]
ERROR: The SUBMIT command failed.

```


What happened? I didn't have a clue. It couldn't find something, but why not? This called for expert advice.

And happily, I was able to find it. There's not a lot written about Proc Groovy. But over at communities.sas.org, there is someone who knows much more than I do. He goes by the username FriedEgg, and he was able to quickly identify the problem, <<https://communities.sas.com/message/102280#102280>>:

It should be part of your standard Java install, but it is probably not included in your environment by default.

```
proc groovy;
```

```
ADD SASJAR="ANT";
```

```
submit;
```

That solved the problem.

There are lots of sources of help for Groovy and Java. There's not much apparent use of Proc Groovy, but I suspect that its use will grow. <<http://www.sascommunity.org>> and <communities.sas.com> are good places to look and ask. With luck, a library of sample Groovy programs will build up at sascommunity.org over time.

MAKING PROC GROOVY RESULTS AVAILABLE TO SAS

There are at least three ways to make the results of a Proc Groovy program available to SAS.

Write to a file

If you are reusing an existing Groovy or Java program that already writes to a file, just let Proc Groovy write the file unchanged, and read it in a subsequent data step.

Use the data step object-oriented interface to Java

FriedEgg has given us a simple example of this. It determines whether the site is in Daylight Savings Time, <<https://communities.sas.com/message/144937#144937>>. Unfortunately, this requires the use of a classpath, and the SAS documentation is so vague that it's virtually useless. I just kept trying different things with classpath until I found something that worked. This example is a slightly modified version of FriedEgg's; I don't actually know what it does, but it does run and create results.

```

1048 filename cp 'c:\temp\test.jar';
1049
1050 proc
1050!   groovy classpath=cp;
NOTE: The ADD CLASSPATH command completed.
1051   submit parseonly;
1052   class DST {
1053     static boolean isDst(String tmz) {
1054       return TimeZone.getTimeZone(tmz).inDaylightTime(new Date())
1055     }
1056   }
1057   endsubmit;
NOTE: The SUBMIT command completed.
1058 quit;

NOTE: PROCEDURE GROOVY used (Total process time):
      real time          0.05 seconds
      cpu time           0.01 seconds

1059
1060 options set=classpath "c:\temp\test.jar";
1061
1062 data _null_;
1063   declare javaobj j('DST');
1064   j.callStaticBooleanMethod('isDst', "EST", Is_DST);
1065   put Is_DST=;
1066   j.callStaticBooleanMethod('isDst', "EDT", Is_DST);
1067   put Is_DST=;
1068 run;

Is_DST=0
Is_DST=0

```

Pass values through macro variables

For passing small amounts of data, this works well. It's a bit of a kludge, but the use of macro variables is well and widely understood, unlike the use of classpath. Proc Groovy supports a special Groovy variable named "exports"; there's a small sample program in the Proc Groovy documentation, and I modified it to export filenames.

```

1263 proc
1263!   groovy;
1264   submit;
1265     filecount = 0
1266     new File('W:/WUSS2012/').eachFile() { file->
1267       filecount = filecount + 1
1268       exports.put('filename' + filecount, file.getName())
1269     }
1270     exports.put('filecount', filecount)
1271   endsubmit;
NOTE: Exporting macro variable "filecount".
NOTE: Exporting macro variable "filename20".
[... lines omitted ]

```

```

NOTE: Exporting macro variable "filename7".
NOTE: The SUBMIT command completed.
1272 quit;

NOTE: PROCEDURE GROOVY used (Total process time):
      real time           0.20 seconds
      cpu time            0.06 seconds

1273
1274 data _null_;
1275     length filename $60.;
1276     filecount = input(symget('filecount'), best.);
1277     putlog 'There are ' filecount 'files';
1278     do i = 1 to filecount;
1279         filename = symget(catt('filename', i));
1280         putlog i= filename=;
1281     end;
1282 run;
[... lines omitted ]
There are 33 files
i=1 filename=.DS_Store

i=33 filename=~$FP_57.docx
NOTE: DATA statement used (Total process time):
      real time           0.08 seconds
      cpu time            0.06 seconds

```

Unfortunately, going the other way doesn't seem to work; a statement like

```
myvalue = exports.get('mymacvar')
```

compiles and runs, but myvalue isn't set to the value of the macro variable mymacvar.

FURTHER READING

A version of this paper will be available online at

http://www.sascommunity.org/wiki/Simple_Proc_Groovy

It is likely to be better and more complete than the soon-to-be-outdated version in the Proceedings.

CONTACT

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

Jack Hamilton
jfh@acm.org:

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.