Adventures in Groovy – Part 12: Learning and Testing Groovy Outside of PBCS

Introduction

For people that are new to Groovy/Java, testing functions that Groovy provides can be a tedious and time consuming process. Learning anything is. Trying to do this with the wrong tools compounds it. I have seen some people give up and walk away from trying to improve applications because they struggle with the Groovy Calculations and the complexity it introduces to go beyond some of the basics, just because they are using a hammer when they need a screwdriver. For example, it is simple to use a documented example and loop through the cells on a form, but to utilize the Groovy/Java objects and methods is the difference between using the default logic and taking Planning to a whole new level. For those of us who are learning, testing simple functions can be very painful inside a Groovy Calculation.

I will by preface saying I am not a Groovy developer. I am learning as I need functionality and I am trying to build a foundation to be as productive as possible. Although Groovy in PBCS doesn't give developers full access to all the Java libraries, much of the logic that is needed to develop new functionality can be tested outside of PBCS. I have found that as I learn more and require more non PBCS related functionality, it is easier to test in the Groovy Console rather than in a PBCS calculation. Some examples are

- string functions like replace, regex, concatenate
- mathematical functions
- other manipulation that require the use of collections

and hash tables

These can be used in looping through grid cells or building evaluation rules on data entered. Hopefully, this is helpful to those learning Groovy.

How To Get Started

Download Java SDK

Before Groovy can be used, Java has to be installed. Most systems already have it. If not, the Java Development Kit can be downloaded and installed. There is information about which version of Groovy and Java are compatible at groovy-lang.org. The Java SDK can be downloaded from Oracle.

Download An Editor

Groovy can be edited in many free and paid programs. Some of them are more robust than others and provide things like automatic code completion, color coding, and more advanced features that aren't likely required at this novice level. They also increase the complexity for those that are completely new to writing JAVA or Groovy. If you are interested in this or need a longer term solution, check out these editors.

- IntelliJ IDEA
- Netbeans IDE
- SlickEdit
- Groovy/Grails Tool Suite

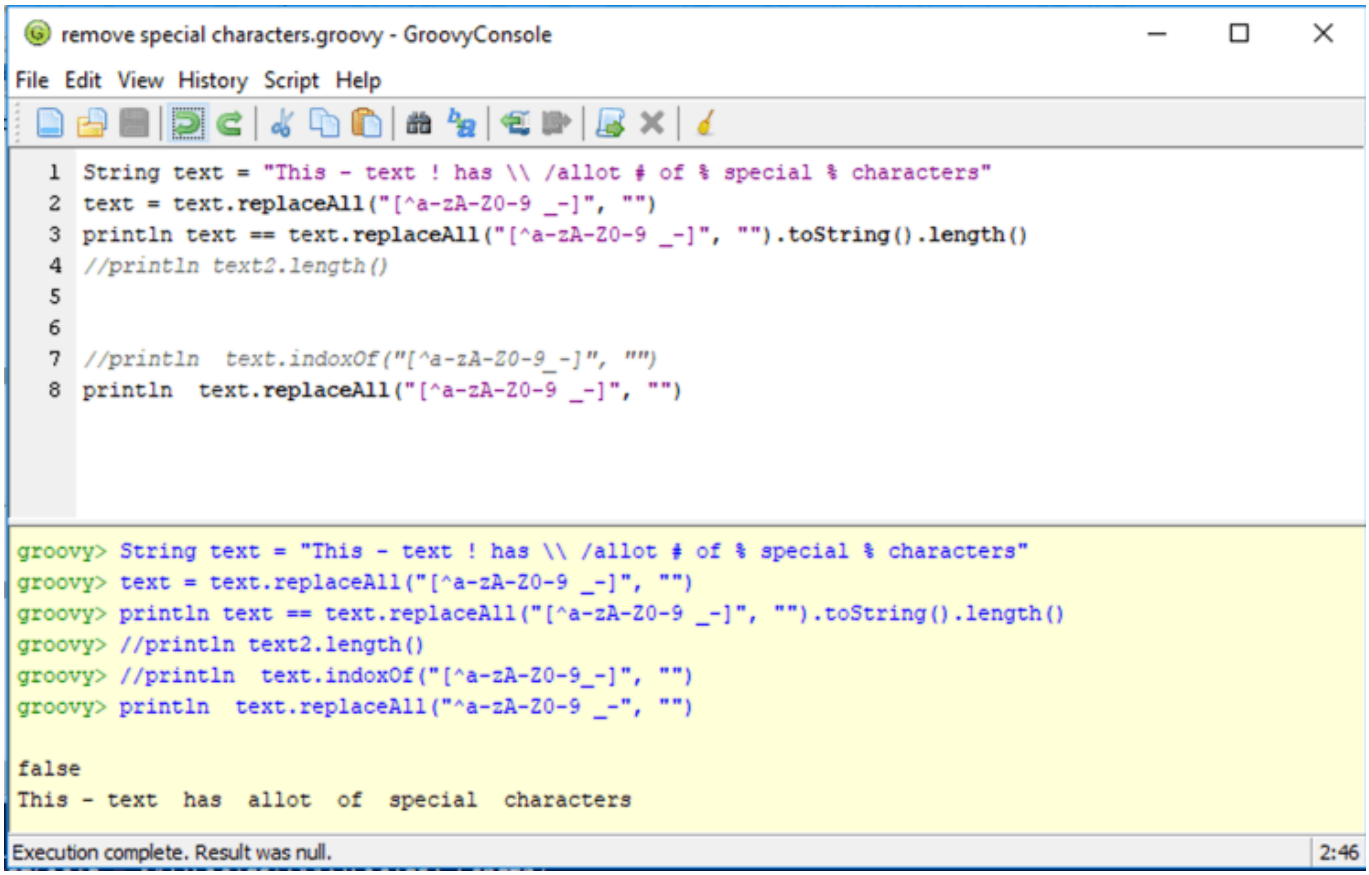
For those who want to just get started with a simple and supported editor to test some basic code, try the Apache Groovy Console. The Windows Installer, the documentation, and the SDK are available to download and install. Once installed, you are ready to go! If you go to your Start menu in Windows, you will see a folder for the version of Groovy installed. In that folder click on *Start GroovyConsole* to open the editor.

Using The Groovy Console

Much of what is done in the Groovy calculations can't be accessed here. We don't have grids, cells, or any of the PBCS methods that we interact with in a Groovy Calculation. Groovy can also access the REST API (outside of Groovy Calculations), which opens up the ability to manage PBCS like EPM Automate. I recently looped through the product catalogue at BestBuy.com and built a hierarchy! This is a whole other beast, but it is worth mentioning.

Before we jump into testing a script, here are a few things that will be helpful using the Groovy Console.

- The editor has two panes. The top pane is where the script is developed and edited. The bottom pane is where the results of the script are displayed when it is executed.
- The toolbar has some common functions. You can open and save your scripts, redo/undo, and execute from icons in this area.



Examples

I find it very helpful as I am learning, to test the logic and the results in this console. Once validated, it will be moved to the PBCS calculation and used appropriately. Here are some examples where it might be useful, and hopefully the separation of where to test what is highlighted.

Regex Example

There was a requirement on a form at a recent client where they wanted to accept input. They used this to setup properties in the HR system. The HR system could not accept some characters, so the ask was to only allow alphanumeric characters, a space, an underscore, and a dash. We had to add validation to the run time prompt, as well as when the data was updated in a form. Not being an expert with regex, I didn't want to test this in a calculation (update calc, run calc, open job console, expand status, toggle between windows, etc).

So, I opened the Groovy Console and tested there. The end result is below, but it was much easier to tweak the regex syntax directly in the console, running it, and seeing the result immediately, in one step. This was easy to see and verify the output was void of any characters that were not allowed. The length could be compared, pre and post character removal, and was used to stop the save of the data.

```
String text = "This - text ! has \\ /allot # of % special % characters"
println text
println text.length()
println text.replaceAll("[^a-zA-Z0-9 _-]", "")
println      text.replaceAll("[^a-zA-Z0-9      _-]", "").toString().length()
println text.length() == text.replaceAll("[^a-zA-Z0-9 _-]", "").toString().length()
```

At this point, I proved out the regex functionality. I can now go back to the Groovy Calculation and use this logic on the variable returned from the PBCS function (whether it be an RTP or a cell value) and remove the invalid characters or test to see if there are any, and act accordingly. This is what it would look like

```
String enteredValue = rtps.RTP_NewEmployee.getEnteredValue();
if(enteredValueAdj.length() == enteredValue.length())
{
def mbUs = messageBundle(["validation.InvalidChars":"You have entered invalid characters.  Only alphanumeric characters, spaces, dashes, and underscores are accepted."])
def mbl = messageBundleLoader(["en" : mbUs])
throwVetoException(mbl,      "validation.InvalidChars", rtps.RTP_NewEmployee)
}
```

Converting Nested Collections

I was building a Data Map override from a POV, and it wasn’t validating because some of the variables were collections that included a nested collection. This whole concept was

completely new to me, and again, I didn't want to have to go through 3-5 steps to see if the result was returning a delimited list of members that the Data Map would accept. Since I had no initial idea how to accomplish this, I searched and found examples that might accomplish what I wanted to achieve. It took 5 to 10 iterations of examples to get to what I wanted and understand how this worked. Updating a script in the Groovy Console, running it, and seeing the results in the same window proved much quicker to find a solution.

In the solution below, I created a variable that replicated the variable that PBCS that was returning (a list). I was able to build out a few lines to eliminate the nested collections and ported this over to my Groovy Calculation.

This proved out that the simple loop below would give me a list I could pass to the Data Map, and was much quicker to solve than trying to do this in PBCS.

```
def orig_list = [10, 20, [1, 2, [25, 50]], ['Groovy']]
def usable_list = []

orig_list.collectNested([]) {
 usable_list << it
 }
println usable_list
println '"' + usable_list.join('","') + '"'
```

The result of the executed script created two lines. At this point, I could use this function in the Groovy Calculation by replacing the orig_list with the object returned from the PBCS function. I used the usable_list in the Data Map.

```
[10, 20, 1, 2, 25, 50, Groovy]
“10”,”20″,”1″,”2″,”25″,”50″,”Groovy”
```

Wrapping Up

These examples are great examples of how we can use a pair of tools to create business logic efficiently. If you are a seasoned java developer, much of this might seem ridiculous to you and question why one would ever use something outside of PBCS. I get it. Now that I know how these two function work, I likely will not use the Groovy Console to write and test this. But, as I continue to learn more and more, being able to do this in something outside of PBCS has proven invaluable, increased my productivity, and significantly reduced my frustration.

If you are learning, or are an experienced Groovy developer, please share your insights with the community and post a comment!