Adventures in Groovy – Part 24: Don’t Be Stingy With Reusable Code

Now that you are knee deep in Groovy, help yourself out and reuse common code. The more you learn, the more efficient you will be and will laugh at some of your initial attempts at your Groovy calculations. If you are like me, you get excited about all the possibilities and learn as you go. When you find better ways to do something, or even preferable ways, you end up with an application with inconsistent representations of the same logic. You are too busy to go back and update all the snippets you have improved, so it stays in a somewhat messy state.

Do yourself a favor and start reusing some of the things you need in every script. When you start doing more in Groovy calculations, the more you can replicate the better. Making some of the code business rule agnostic will make you more productive and will create a consistent approach to calculation strategy. An example of this would be variables for the dimensions in the POV. Or, maybe a common map used to push data from WFP to the P&L. Regardless of the use, as soon as you see things that will be duplicated, put them in a script. Scripts can be embedded in Groovy calculations just like regular Business Rules.

Header Script

This is an example of something that I find useful for every Groovy calculation I am writing. It accomplished a couple things.

1. It stores the forecast months so data maps, smart pushes, Essbase calculations, and grid builder

functions, can be dynamically execute on all months for a budget and the out months for the forecast based on the scenario selected.
2. It gets the numeric value for the current month to be used in other localized calculations.
3. It creates a calendar object to get the current time for logging performance.
4. The parameters are logged to the job console.

```
/*RTPS: {RTP_Month}*/

def Month = rtps.RTP_Month.toString().replaceAll("\"", "")
def MonthFix = rtps.RTP_Month.toString()
def months =
['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','
Nov','Dec']
def actualMonths = []
def forecastMonths = []

months.eachWithIndex {item, index ->
  if(index > months.findIndexOf{it == Month})
    forecastMonths <<  item
  else
    actualMonths <<  item
}

def actualMonthsFix = """\"${actualMonths.join('", "')}\""""
def  forecastMonthsFix  =  """\"${forecastMonths.join('",
"')}\""""
def useMonths = []

if(operation.grid.pov.find{it.dimName
=='Scenario'}.essbaseMbrName.toString().toLowerCase().contains
('forecast'))
  useMonths = forecastMonths
else
  useMonths = months

//Get time for logging perforance
Calendar calendar = Calendar.getInstance()
calendar.setTimeInMillis(currentTimeMillis())
```

```
// Get the numeric value for the month (Jan=0)
int monthNumber = Calendar.instance.with {
  time = new Date().parse( "MMM", Month )
  it[ MONTH ]
}

println "The current month number is $monthNumber"
println "The current month is $Month"
println "The current month (Fix) is $MonthFix"
println "The actual months (list) are ${actualMonths}"
println "The actual months (string) are ${actualMonthsFix}"
println "The forecast months (list) are ${forecastMonths}"
println "The forecast months (string) are ${forecastMonthsFix}"
```

Conversion Table

If the application moves data from more detailed plan types to a consolidated P&L, it likely has conversions. This is an example of an account map from a Gross Profit plan type to a P&L plan type. How it is used will be explained in a later submission, but it is used in every calculation that synchronizes data from the GP to the P&L cube. Therefore, it is a great candidate to have in a shared library (a script) so it can be maintained in one place.

```
def acctMap = ['Cases':'Units',
               'Sales':'42001',
               'COGS':'50001',
               'Tax':'50015',
               'Chargeback':'56010',
               'Investment Gains':'50010',
               'Writeoffs':'56055',
               'Growth Attributed to New Products':'56300',
               'Samples':'56092'
                ]
```

Sharing Scripts

Just as scripts can be embedded in regular business rules, the

exact same syntax is used in Groovy calculations. Assume the header script above is called Common Groovy and sits in an application named FinPlan in the GP plan type. The inclusion of the script into any Groovy calculation would be done by inserting the following text.

```
%Script(name:=" Common Groovy ",application:="FinPlan",plantype:="GP")
```

Conclusion

These are just examples to get you thinking about how you can reduce the headaches down the road when your Groovy calculations need maintained. It can easily be expanded to include common POV members and other common code. Be mindful of how global this is as not all POV members are in all forms. You might find it useful to have a script that is shared everywhere and a few others shared for like calculations or forms. You aren’t limited to only including one shared script.

If you have a snippet you are using in all your Groovy calculations, share it with the community. We always like to get feedback and learn from each other.