A Totally Groovy Essbase Hands on Lab

Presented By
Dave Schleis, Insum Solutions and Joe Aultman, interRel Consulting

Exercise 1 – Hello World (5 min.)

1. Open the Groovy Console
2. Create your rendition of the requisite "Hello World" program.
3. Select **Script -> Run (Ctrl-R)**

Exercise 1.5 – Assert yourself (5 min.)

1. Within the Groovy Console type something like:

```
a = 2
b = 3

assert a + a == 4

assert a + b == 4
```

2. Run the script.
3. Note the wonderful detail that the Groovy assert provides.
4. Make some other assertions.

Exercise 2 – Keep calm and assert yourself (5 min.)

5. Within the Groovy Console type (or copy and paste):

```
a = [1,2,3]
b = [2,4,6]
a.each {
  println it * 2
}
g = {println it * 3}
a.each(g)

c = []
a.each { aNum ->
  c << aNum * 2
}

assert c == b
assert b == a * 2
```

6. Run the script.
7. Experiment with println, each{} and assert.

Exercise 3 – Here come the JAPI (5 min.)

1. Within the Groovy Console type (copy/paste) the following:

```
import com.essbase.api.session.IEssbase

essHome = IEssbase.Home.create(IEssbase.JAPI_VERSION)

essSvr = essHome.signOn('epm_admin', 'Passw0rd', false, null,
'http://kscope:19000/aps/JAPI', 'localhost')

essApp = essSvr.getApplication('Sample')

essCubes = essApp.getCubes()

essCubeArray = essCubes.getAll()

essCubeArray.each {
   println it.name

}

essSvr.disconnect()
essHome.signOff()
```

2. Check out this script.
3. What is the closure going to do?
4. Run the script.
   The output should be a listing of the cubes within the Sample application.
5. Modify the script to be more compact and Groovyesque.
6. Run the script.

Exercise 4 – Did I dimension that Groovy rocks? (5 min.)

1. Within the Groovy Console type (copy/paste) the following:

```
import com.essbase.api.session.IEssbase

essHome = IEssbase.Home.create(IEssbase.JAPI_VERSION)
essSvr = essHome.signOn('epm_admin', 'Passw0rd', false, null,
'http://kscope:19000/aps/JAPI', 'localhost')

essSvr.getApplication('Sample').cubes.all.each { essCube ->
    println()
    println "${essCube.name}:"
    println '-' * ( essCube.name.size() + 1 )
    essCube.dimensions.all.each {
        println it.name
    }
    essCube.clearActive()
}

essSvr.disconnect()
essHome.signOff()
```

2. Again, check out this script. The commands clearly tell you what they are doing. No fluff, just stuff. What will the output be?
3. Note that we are not using the default "it" variable name when stepping through the cubes.
4. Run the script.
   The output should be a listing of the cubes in the Sample application, along with their dimensions.
5. Modify the script to display the dimension names indented in a stair step fashion based on their position.

```
Basic:
------
 Year
  Measures
      …
```

6. If you have time, modify the program to display all of the output centered in a 40-character column.

Exercise 5 – Just get down to business with Egress (5 min.)

1. Using your favorite text editor (shortcuts on the desktop), create a new file and add the following:

```
import com.hyperionaddict.egress.Egress

Egress.withServer( new KS16SignOn() ) { svr ->
    svr.withApplication( 'Sample' ) { app ->
        app.eachCube() { cube ->
            println()
            println "${cube.name}:"
            println '-' * ( cube.name.size() + 1 )
            cube.dimensions.each {
                println it.name
            }
        }
    }
}
```

2. Save the file as ***SampleDims.groovy*** in the ***C:\Kscope16\GroovyCode*** folder.
3. Compare this code with the last exercise. Notice how Egress make it even easier for you to get down to the business of getting things done.
4. Open a command window using the shortcut on the desktop.
5. Run your program using the command "**groovy SampleDims.groovy**"
   The output should be the same as in the previous exercise
6. Modify the program to create a **list** of the dimension names within the cube, then iterate through the list to print the names.
   Hint:

```
myList = []
myList << thingToAddToList
```

Exercise 6 – Take command (5 min.)

1. Using your favorite text editor (shortcuts on the desktop), create a new file and add the following:

```
def cli = new CliBuilder(usage: 'CalcIt.groovy -[asmdh] -x Xval -y Yval')

cli.with {
   h longOpt: 'help', 'Show usage information'
   a longOpt: 'add', 'Xval + Yval (default)'
   s longOpt: 'subtract', 'Xval - Yval'
   m longOpt: 'multiply', 'Xval * Yval'
   d longOpt: 'divide', 'Xval / Yval'
   x longOpt: 'X', required: true, args:1, 'Xval'
   y longOpt: 'Y',  required: true, args:1, 'Yval'
}

def options = cli.parse(args)
if (options.h) {
  cli.usage()
}
x = options.x.toInteger()
y = options.y.toInteger()

if (options.s) println x - y
else if (options.m) println x * y
else if (options.d) println x / y
else println x + y
```

2. Save the file as ***CalcIt.groovy*** in the ***C:\Kscope16\GroovyCode*** folder.
3. Open a command window using the shortcut on the desktop.
4. Run your program using the command "**groovy CalcIt.Groovy –x 2 –y 3**"
5. Run the program several times using different arguments
6. If you have time, modify the program to handle things gracefully when no arguments are given, or when x or y are not provided.

Exercise 7 – Command and control (10 min.)

1. Using your favorite text editor create a new file and add the following:

```
import com.hyperionaddict.egress.Egress

def cfg = new ConfigSlurper().parse(new File('credentials.groovy').toURL())

assert cfg != null

def cli = new CliBuilder(usage: "groovy ${getClass().simpleName} [options]
scriptName...\n Use empty string \"\" to indicate the default calc.", header:
'(Application & Database parameters required)')

// Builder paradigm allows easy configuration of parameters.
cli.with {
    u args: 1, longOpt: 'user', 'username for Essbase'
    p args: 1, longOpt: 'pass', 'password for Essbase'
    s args: 1, longOpt: 'server', 'Essbase server'
    a required: true, args: 1, longOpt: 'app', 'Essbase application'
    d required: true, args: 1, longOpt: 'db', 'Essbase database'
}

def options = cli.parse(args)

if (!options) {
    return
}
if (options.arguments().size() < 1) {
    println 'error: missing scriptName'
    cli.usage()
    return
}

// Elvis operator selects the provided command line option if there is one and
defaults to the config file setting if there is not an option provided.
server = options.s ?: cfg.svr
user = options.u ?: cfg.usr
pass = options.p ?: cfg.pw

Egress.withServer(server, user, pass) { svr ->

svr.withCube(options.a, options.d) { cube ->
        options.arguments().each {
            if (it == '') {
                print "Executing default calc in ${options.a}.${options.d}..."
                cube.calculate()
            } else {
                print "Executing ${options.a}.${options.d}.${it}..."
                cube.calculate(false, it - '.csc')
            }
            println "Done!"
        }
    }
}
```

2. Save the file as ***RunCalc.groovy*** in the ***C:\Kscope16\GroovyCode*** folder.
3. Examine the code. Read the comments to make sure you understand what is going on at each step. Why do you think it's called the Elvis operator?
4. Run your program using the command **groovy RunCalc.groovy** to see usage.
5. Run your program using the command **groovy RunCalc.groovy -a Sample -d Basic ""**

You will see a lot of scary looking stuff, but the very last line should read:
"Executing default calc in Sample.Basic...Done!"

6. Challenge: Modify script & config file to allow use of withServer(essSignOn) instead.

Exercise 8 – Egress to Impress (15 min.)

1. Make a copy of the Sample application:
2. Using your favorite text editor create a new file and add the following:

```
import com.hyperionaddict.egress.Egress

Egress.withServer( new KS16SignOn() ) { svr ->
    // Remove copy of Sample app, if one already exists
    if (svr.getApplicationOrNull('SampCopy')) {
        svr.withApplication( 'SampCopy' ) { app ->
           app.delete()
        }
    }
    // make sure it's not there
    assert (svr.getApplicationOrNull('SampCopy')) != null

    // Create copy of Sample app
    svr.withApplication( 'Sample' ) { app ->
       app.copy( 'SampCopy' )
    }

    // make sure it is there
    assert (svr.getApplicationOrNull('SampCopy')) != null
}
```

3. Save the file as ***CopySamp.groovy*** in the in ***C:\Kscope16\GroovyCode***.
4. Run the program using the command **groovy CopySamp.groovy**.
   There should be no output
5. Create a program to loop through Market members and check for an associated Population member:
6. Using your favorite text editor create a new file and add the following:

```
import com.hyperionaddict.egress.Egress
import com.essbase.api.metadata.IEssAttributeQuery
import com.essbase.api.datasource.IEssCube.EEssRestructureOption

Egress.withServer( new KS16SignOn() ) { svr ->

  svr.withCube('SampCopy', 'Basic') { cube ->
       // Loop through Market members and check for an associated Population
member (should check out)

       cube.withMemberSelection { sel ->
           assert sel.members != null
           sel.executeQuery('<OutputType Binary <SelectMbrInfo (MemberName)',
'@RELATIVE("Market", 0)')
           sel.members.each { mbr ->
               assert mbr.name != null
               results = cube.executeAttributeQuery { qry ->
                   qry.set(IEssAttributeQuery.MBR_TYPE_BASE_MEMBER,
IEssAttributeQuery.MBR_TYPE_ATTRIBUTE_MEMBER)
                   qry.setInputMember(mbr.name)
                   qry.setAttributeValue(IEssAttributeQuery.OP_EQ, 'Population')
               }
```

```
                println "${mbr.name}: ${(results.size() == 1) ? 'check' :
'NOOOOOOOO!'}"
            }
        }
    }
}
```

7. Save the file as ***AuditAttributes.groovy*** in the in ***C:\Kscope16\GroovyCode***.
8. Take a close look at the assertions, do they seem correct?
9. Run the program using the command **groovy AuditAttributes.groovy**.
   The output should list each Market member followed by "check".
10. Disassociate the California market from its Population attribute by entering the following code into the Groovy Console and then running it:

```
import com.hyperionaddict.egress.Egress
import com.essbase.api.metadata.IEssAttributeQuery
import com.essbase.api.datasource.IEssCube.EEssRestructureOption

Egress.withServer( new KS16SignOn() ) { svr ->
    assert svr == null
    svr.withCube('SampCopy', 'Basic') { cube ->
        cube.withOutline { otl ->
            otl.disassociateAttributeMember(otl.findMember('California'),
otl.findAttributeMembers('33000000', '')[0] )
            otl.save(EEssRestructureOption.KEEP_ALL_DATA)
        }
    }
}
```

   There should be no output.
11. Open EAS and review the outline of SampCopy.Basic to note that California is no longer associated with its population.
12. **CLOSE THE OUTLINE**.
13. Run your **AuditAttributes.groovy** program again.
    The output should list each Market member followed by "check", except for California.
14. Disassociate another population from its market by replacing 'California' in the script in the Groovy Console.
15. Run your **AuditAttributes.groovy** program again.
    Is the output what you expected?
16. Clean up:
17. Enter the following into the Groovy Console

```
import com.hyperionaddict.egress.Egress

Egress.withServer( new KS16SignOn() ) { svr ->
    // Remove copy of Sample app
    svr.withApplication( 'SampCopy' ) { app ->
        app.delete()
    }
}
```

18. Run the Script.
    There should be no output

Exercise 9 – Load but verify (15 min.)

1. Parse the data and add up totals:
2. Within the Groovy Console type (copy/paste) the following:

```
// Set up variables to accumulate data from file
def totalSales, totalCogs, totalMarketing
(totalSales, totalCogs, totalMarketing) = [0, 0, 0]

// Open file, get content, discard first line, filter lines, iterate
new File('Calctxt0.txt').readLines().tail().findAll{ it =~ /"Actual"/ }.each {
line ->
    // split the line into an array based on the appropriate character
    parts = line.split('"')
    assert parts != null
    dims = parts[1, 3, 5, 7]
    data = parts[-1].trim().split(' ')
    data = data.collect{ ( it == '#Mi' ) ? 0.0 : it.toDouble() }
    (totalSales, totalCogs, totalMarketing) = [ totalSales + data[0], totalCogs
+ data[1], totalMarketing + data[2] ]
}

// Output results
println()
println 'From data file:'
println "Sales: $totalSales"
println "Cogs: $totalCogs"
println "Marketing: $totalMarketing"
```

3. Run the script in the Groovy Console.
   The output should list the totals for the three columns of interest.
4. Look more deeply into what is going on in the code by pasting this into the Groovy Console:

```
// Open file, get content, discard first line, filter lines, iterate
dataFile = new File('Calctxt0.txt')
dataArray = dataFile.readLines()
println "dataArray[0..3] = ${dataArray[0..3]}\n"

dataArrayTail = dataArray.tail() // remove the first element of each of the
array lines
println "dataArrayTail[0..3] = ${dataArrayTail[0..3]}\n"
println "dataArrayTail[12] = ${dataArrayTail[12]}\n"

dataArrAct = dataArrayTail.findAll{ it =~ /"Actual"/ }
println "dataArrAct[0..3] = ${dataArrAct[0..3]}\n"
println "dataArrAct[12] = ${dataArrAct[12]}\n"

partsArr = dataArrAct[12].split('"')
println "partsArr=$partsArr\n"

partDims = partsArr[1, 3, 5, 7]
println "partDims=$partDims\n"

partData = partsArr[-1].trim().split(' ')
println "partData=$partData\n"

partData2 = partData.collect{ ( it == '#Mi' ) ? 0.0 : it.toDouble() }
println "partData2=$partData2\n"
```

5. Load data to cube and query to verify it loaded:
6. Using your favorite text editor create a new file and add the following:

```
import com.hyperionaddict.egress.Egress
import com.essbase.api.datasource.IEssOlapFileObject

// Model query with a simple tab-delimited multi-line string
query = """\
       Year   Product       Market Scenario
Sales
COGS
Marketing
"""

Egress.withServer( new KS16SignOn() ) { svr ->
    svr.withCube('Sample', 'Basic') { cube ->
        cube.clearAllData()
        // Try commenting out the next line and see what happens...
        cube.beginDataload(null, IEssOlapFileObject.TYPE_RULES, new
File('Calctxt0.txt').canonicalPath, IEssOlapFileObject.TYPE_TEXT, false, 0)
        cube.calculate() // Runs default calc
        cv = cube.openCubeView('default') // You can call it anything
        grid = cv.getGridView()
        // Use query string content to set grid size
       grid.setSize( query.readLines().size(),
query.readLines()[0].split('\t').size() )
        // Parse string to put values in the grid
        query.eachLine(0) { line, lnum ->
            parts = line.split('\t')
            parts.eachWithIndex { p, i ->
                grid.setValue(lnum, i, p)
            }
        }
        op = cv.createIEssOpRetrieve()
        cv.performOperation(op)
        println()
        println 'From Essbase:'
        // Have to check for #Missing before trying to pull data from the grid.
(getDoubleValue on #Missing throws an exception)
        if (grid.getStringValue(1, 1) == '#Missing') {
          println 'Sales: 0'
        else
          println "Sales: ${grid.getDoubleValue(1, 1)}"
        }

        println "Cogs: ${ if (grid.getStringValue(2, 1) == '#Missing') '0' else
grid.getDoubleValue(2, 1)}"

        println "Marketing: ${(grid.getStringValue(3, 1) == '#Missing') ? 0.0 :
grid.getDoubleValue(3, 1)}"
    }
}
```

7. Save the file as ***ValidLoad.groovy*** in the in ***C:\Kscope16\GroovyCode***.
8. Examine the code.
   Note how commands are strung together in an almost sentence-like structure to efficiently and descriptively get the job done.
9. Run the program using the command **groovy ValidLoad.groovy**.

The output should match what was seen in the Groovy Console when adding up the data from the file.

10. Add the code from the first half of the exercise to your ValidLoad.groovy program and run the program again.
11. If you have time, add code to alert you in the event of a discrepancy between the file parsing total and the Essbase query total.

Exercise 10 – Sort it all outline (15 min.)

1. Using your favorite text editor create a new file and add the following:

```
import com.hyperionaddict.egress.Egress
import com.essbase.api.metadata.IEssAttributeQuery
import com.essbase.api.datasource.IEssCube.EEssRestructureOption

Egress.withServer( new KS16SignOn() ) { svr ->
    // Remove copy of ASOcopy app, if one already exists
    if (svr.getApplicationOrNull('ASOcopy')) {
        svr.withApplication( 'ASOcopy' ) { app ->
           app.delete()
        }
    }

    assert svr.getApplicationOrNull('ASOcopy') == null

    // Create copy of ASOsamp app
    svr.withApplication( 'ASOsamp' ) { app ->
       app.copy( 'ASOcopy' )
    }

    assert svr.getApplicationOrNull('ASOcopy') != null

    svr.withCube('ASOcopy', 'Sample') { cube ->    // Could have used
svr.withOutline(app, db), except the cube object is needed below for attribute query.
        cube.withOutline { otl ->
            println()
            println 'Sorting "Products" alphabetically:'
            // Iterate through all members of the "Products" dimension.
            otl.findMember('Products').eachDescendant(true) { mbr ->
                println mbr.name
                // Sort member's children by name. (Lev 0 members have no children.
Challenge question: What happens? Why?)
                // Reverse the order, then move each into the first slot under its
parent.
                // Going in reverse order pushes the previously moved members down the
outline as siblings are added above.
                // Resulting order in outline is ascending by name.
                mbr.childMembers.all.sort{it.name}.reverse().each { child ->
                    otl.moveMember(child, mbr, null)
                }
                // Challenge: Modify so that "Other" stays below its siblings, instead of
moving between them.
            }
            println()
            println 'Sorting "Stores" by Square Footage:'
            otl.findMember('Brick & Mortar').eachDescendantAtLevel(1) { mbr ->
                println mbr.name
                // Sort first by store name, flip the results, then sort again by
attribute value.
                // Resulting order in outline is descending by attribute value, ascending
by member name when the attribute is the same.
                members =
mbr.childMembers.all.sort{it.name}.reverse().sort{cube.getAssociatedAttributes(it.name,'S
quare Footage')[0][0].toInteger()}.each { child ->
                    otl.moveMember(child, mbr, null)
                }
                // Challenge: Modify to sort by Store Manager instead of member name when
the Square Footage is the same.
            }
            otl.save(EEssRestructureOption.KEEP_ALL_DATA)
```

```
        }
    }
    // making a script interactive
    javax.swing.JOptionPane.showMessageDialog null, "Review then close outline. ASOcopy
will be deleted."

    // Remove copy of ASOsamp app
    svr.withApplication( 'ASOcopy' ) { app ->
       app.delete()
    }

    assert svr.getApplicationOrNull('ASOcopy') == null

}
```

2. Save the file as ***SortOutline.groovy*** in the in ***C:\Kscope16\GroovyCode***.
3. Examine the code carefully, to understand what each line is doing.
4. Run the program using the command **groovy SortOutline.groovy**.
   The output should consist of a listing of products and stores before they were sorted.
5. Open up EAS and see that things have been sorted.
6. BE SURE TO CLOSE THE OUTLINE!
7. When you have finished verifying that the sorting has taken place, click on the OK button in the Message popup to finish the program.
8. Modify your script to meet the challenges and run again to see the difference:
9. Challenge 1: Modify the "Product sort" so that "Other" stays below its siblings, instead of moving between them.
10. Run the program then check to see that things sorted properly.
11. Challenge 2: Modify the "Store sort" to sort by Store Manager instead of member name when the Square Footage is the same.
12. Run the program then check to see that things sorted properly.

Exercise 11 – It would be great, if you would evaluate (3 min.)

1. Fill out the course evaluations.
2. Please, let us know what you thought. Really.