Adventures in Groovy – Part 42: Using Scenario Start and End Periods Through Metadata Queries Provides Awesome Functionality

Identifying the months to calculate plan and forecast has always been a delicate balance between performance and maintenance. Having a calculation for each required duplication of forms or calculating more than what was required. Calculations required more complicated if logic than what it should have or (I hate when I see this) a calculation that just does both forecast and plan regardless of what needs calculated.

Groovy has opened up a lot of options. We can identify if plan or forecast is in the form and dynamically build the Essbase calculation, which is what I have started doing. This, combined with only calculating on the edited cells, gets you part of the way there. It doesn't help with global calculations or calculations when a single month will impact all months, like when there are balance walk forwards.

Groovy provides us the ability to get member properties, which I have discussed before. The Scenario dimension has unique properties, like the start month, start year, end month, and end year, which is what creates the security on the periods and years for each scenario. Once these are identified, the possibilities are endless. They can be used to create gridbuilders, they can be used to dynamically fix on the correct periods in Essbase calculations, or they can be used to do other date specific things.

Groovy Calculation and API

At this point, you are probably very well aware of the operation class, which has an application sub class. The application subclass has many functions within it that are useful. One, is the ability to create a dimension object. From there, a member object can be created. The following is a very simple example to hold a scenario member in a member object that can be used in many ways.

```
Member mbrScenario = operation.application.getDimension('Scenario').getMember('Plan')
```

If the scenario is a run time prompt, it would look like this. Assume the RTP is named rtpScenario.

```
Member mbrScenario = rtps.rtpScenario.getMember()
```

When the member object is initiated, all the properties of the specific member are available. You may want to review Adventures In Groovy – Part 11: Accessing Metadata Properties before continuing. The same concept is used here, but the properties returned are slightly different. The four we want to access are

- Start Period
- End Period
- Start Year
- End Year

These can be stored in variables or used within calculations or other logic. There may be a need to convert the month member to a long month or a number. The year may also be converted to remove the FY and prepend 20 so it can be used in date manipulation. println mbrScenario.toMap() produces the following, so the full list of properties is a little different than what was produced in Part 11.

Log messages :

```
{Parent=Scenario, Two Pass Calculation=false, Process Management Enabled=true, Old Name=Plan, Formula=<none>, End Year=FY20, UDA=, Aggregation (GTech)=+, Solve Order (GTech)=0, Essbase Name=Plan, UUID=91f18d30-fc37-4e0b-bc9a-99668194e793, Member=Plan, Data Storage=never share, Hierarchy Type=none, Allow Upper Level Entity Input=false, End Period=Dec, Start Period=Jul, Aggregation (ProdRev)=+, Include BegBal=true, Data Storage (GTech)=never share, Formula Description=<none>, Data Id=2573227211374885, Data Storage (ProdRev)=never share, Start Year=FY18, Data Type=unspecified, Formula (GTech)=<none>, Old Unique Name=<none>, Formula (ProdRev)=<none>}
```

Since this is a Groovy map, a type of Groovy collection, any of the elements can be accessed by referencing the element key.

```
println mbrScenario.toMap()["Start Year"].toString()
```

Multi Year Example

Since many applications will span over multiple years, there is some additional logic that has to be built to loop through the months. If the user is prompted for a start period/year and end period/year, you may need to validate that the date range is within the start/stop periods on the scenario, which also is easier to evaluate if these are converted to dates.

[membership level=”0″]

[/membership]
[membership]

Building The Date Range

Creating a map with the year and the list of months is a great way to store the date range. It can be used to build lists for gridbuilders that read data (parameters are multiple rows/columns by year with periods as a list object), creating Essbase calculations by year, and headers for grids that are submitting data (which are string arrays).

Expanding on what was previously discussed, this will convert the year to a valid year by removing FY and create a date object that stores the start and end dates in variables. Notice the start date uses a day of 1 and the end date uses a day of 2. This is important later on.

```
Member mbrScenario = operation.application.getDimension('Scenario').getMember('Plan')
Date scenarioStart = Date.parse("yyyy-MMM-dd", "20${mbrScenario.toMap()["Start Year"].toString().substring(2)}-${mbrScenario.toMap()["Start Period"].toString()}-01")
Date scenarioEnd = Date.parse("yyyy-MMM-dd", "20${mbrScenario.toMap()["End Year"].toString().substring(2)}-${mbrScenario.toMap()["End Period"].toString()}-02")
```

Now that we have two dates, the next step is to create the map to hold the results. This map will look like similar to this for a date range when Jul,19 through Dec, 20 is selected.

```
['FY19',['Jul','Aug','Sep','Oct','Nov','Dec']
, ['FY20',['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']
```

This is created by looping through the two dates by month. This example uses a while loop and will iterate through each month WHILE the iteration date (current) is before the end date. If the scenarioStart and scenarioEnd dates had the say day, this would exclude the last month because they would be the same. This also uses the calendar class to increase the date by month, which isn't available in the date class.

```
// Create the variables needed
// PeriodRangeMap holds the years/months
// calendar and current are used in the iteration process
Map<String, List<String>> periodRangeMap = [:]
Calendar calendar = Calendar.getInstance()
Date current = scenarioStart

while (current.before(scenarioEnd)) {
   // Set the calendar variable to the date of the current variable
  calendar.setTime(current)
  // If this is a new month, create a null map element to hold the new year and months
  if(periodRangeMap["FY${current.format("YY")}".toString()] == null){
     periodRangeMap["FY${current.format("YY")}".toString()] = []
  }
  // Append the month to the appropriate year
   periodRangeMap["FY${current.format("YY")}".toString()] << current.format("MMM").toString()

  // Increase the calendar variable by one month
  calendar.add(Calendar.MONTH, 1);
  // set the current variable to the next month's date
  current = calendar.getTime();
}
```

If we print the periodRangeMap to the job console using a println, assuming the start month is Jul of 18 and the end month is Dec of 20, this will be displayed with the exception that line returns are added to make it readable.

```
Log messages :
{
FY18=[Jul, Aug, Sep, Oct, Nov, Dec],
FY19=[Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov,
Dec],
FY20=[Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov,
Dec]
}
```

The periodRangeMap can then be used to loop through and create the lists and maps used in gridbuilders and datamaps.

[/membership]

Last Call

The ability to directly access the start and stop period/year really gives partners a new thought process to implementation options where these periods are critical. It is very easy to access, very easy to implement, and extremely useful. If you have an idea of how to use these dates, share them with the community so we can all benefit. Hope to see an idea soon!