Working With Planning Formula Expressions

Most of us know that there is a button in the calc rule editor that allows us the ability to select a smart list and the smart list entry. It will add something [[*smartlist name*.*smartlist entry*]]. If this is new to you, what it does is replace reference the smart list and replaces it with the numeric value that exists in Essbase. The beauty of this is that it is dynamic, so if the smart list is changed in any way, you don't have to go into your rules and replace change the index values for the smart list entries to match. Guess what, there are more!

Well, don't I feel like the F*@$& idiot, to pull a quote from A Few Good Men.

What Is A Planning Formula Expression

As described above, it is an expression that allows you to get valuable information dynamically about artifact properties in a Planning application. The following formula expressions currently exist.

- SmartLists
- Dimensions
- Planning User Variables
- Periods
- Scenarios
- Cross-References
- Workforce Cube Year to Date
- Get ID for String

SmartLists

I already discussed the Smart List, but here is an example if this is new to you. The calculation manager syntax is [[SLName.entryname]].

```
FIX (Mar, Actual, Working, FY15, P_000, "111")
   "Product Channel" = [[Channel.Retail]] ;
ENDFIX
```

Which would return something like this.

```
FIX (Mar, Actual, Working, FY15, P_000, "111")
   "Product Channel" = 2 ;
ENDFIX
```

Dimensions

The dimension expressions are not all that useful unless you are building calculations that might go across applications that have different names for the 6 required dimensions, plus currency. Using the following dimension tags, the customized name will be returned when they are added to the dimension expression. The syntax used for this function is [[Dimension("DIM_NAME_ENTITY")]].

- DIM_NAME_PERIOD
- DIM_NAME_YEAR
- DIM_NAME_ACCOUNT
- DIM_NAME_ENTITY
- DIM_NAME_SCENARIO
- DIM_NAME_VERSION
- DIM_NAME_CURRENCY

An example would look like this. This runs a calc dim on whatever your account dimension is.

```
CALC DIM([[Dimension("DIM_NAME_ENTITY")]]);
```

In this application, Entity is named Entity, so the above script returns:

```
CALC DIM ("Entity");
```

If the entity dimension was named Cost Center, it would return:

```
CALC DIM ("Cost Center");
```

Planning User Variables

Planning user variables return the user variable's member. This can be pretty useful if you have variables that are used to do things like fix on their area of a hierarchy. These can be gathered through run time prompts if they exist in the POV, but that isn't always the case. You might use them to show the products, for example, that are under a user variable, in the rows, in which case without Groovy, it can't be passed in RTPs. The calculation manager syntax is [[PlanningFunctions.getUserVarValue("xyz")]].

An example where the user variable is used to run a calculation might look like this.

```
FIX (Feb, Actual, Working, E_000, @RELATIVE([[PlanningFunctions.getUserVarValue("Product View")]],0) )
   Revenue = Units * Cost;
ENDFIX
```

Period Functions

Period(periodName) returns the specified period. The options for this function are and the calculation manager syntax is [[Period("FIRST_QTR_PERIOD")]].

- FIRST_QTR_PERIOD
- SECOND_QTR_PERIOD
- THIRD_QTR_PERIOD
- FOURTH_QTR_PERIOD
- FIRST_PERIOD
- LAST_PERIOD

This example:

```
FIX ( Mar, Actual, Working, P_000, "6100", FY15 )
   "120" =[[Period("FIRST_QTR_PERIOD")]];
ENDFIX
```

would return a script like this

```
FIX (Mar, Actual, Working, P_000, "6100", FY15)
   "120" = "Mar";
ENDFIX
```

The NumberofPeriodsInYear returns the number of periods in the year and NumberofYears returns the number of years in the application. The calculation manager syntax for this is

- [[NumberOfPeriodsInYear]]
- [[NumberOfYears]]

The following example

```
FIX (Mar, Actual, Working, P_000, "6100", FY15)
   "120"=[[NumberOfPeriodsInYear]];
   "120"=[[NumberOfYears]];
ENDFIX
```

would produce this.

```
FIX (Mar, Actual, Working, P_000, "6100", FY15)
   "120"=12;
   "120"=9;
ENDFIX
```

Scenarios

This one is my favorite ones. I have been using Groovy to get these not knowing they existed. These allow the reduction of if statements and improved performance. We can get the open periods. If this go across years, then my groovy solution probably comes back into play. For ranges that include one year, or even two could be handled, this offers great functionality. The options available are

- Start Year
- End Year
- Start Month
- End Month

The calculation manager syntax is as follows.

- [[getStartYear(“ScenarioName”)]]
- [[getEndYear(“ScenarioName”)]]
- [[getStartMonth(“ScenarioName”)]]
- [[getEndMonth(“ScenarioName “)]]

A use case would look something like this. Assume{rtpScenario} is a run-time prompt variable of type member with a default value of “Actual”:

```
FIX({rtpScenario},
[[getStartYear({rtpScenario})]]:[[getEndYear({rtpScenario})]],

[[getStartMonth({rtpScenario})]]:[[getEndMonth({rtpScenario})]
])
      FIX ( Working, P_000, "111")
        "5800" = 5500;
      ENDFIX
ENDFIX
```

This would build out the following calculation

```
FIX ("Actual", "FY10" : "FY18", "Jan" : "Dec")
    FIX (Working, P_000, "111")
      "5800" = 5500;
    ENDFIX
ENDFIX
```

If your open range consisted of two years, you could do something like this

```
FIX({rtpScenario},[[getStartYear({rtpScenario})]],[[getStartMo
nth({rtpScenario})]]:”Dec”)
      FIX ( Working, P_000, "111")
         "5800" = 5500;
      ENDFIX
```

```
ENDFIX
FIX({rtpScenario},[[getEndYear({rtpScenario})]],"Jan:[[getEndMonth({rtpScenario})]]
      FIX ( Working, P_000, "111")
        "5800" = 5500;
      ENDFIX
ENDFIX
```

Cross-References

This function comes in a few flavors but does something pretty awesome. How it works might change your naming convention a little, or make it more consistent anyway. What id does is generate a cross dimensional reference to our default members, like No Product. The syntax is CrossRef(accountName, prefix, true) but the last two parameters are optional. If you use CrossRef(“Revenue”), it would produce the following, assuming your 6 required dimensions and a product dimension.

```
"BegBalance"->"No Scenario"->"No Version"->"No Entity"->"No Product"->"Revneue";
```

I can change my prefix by adding the second parameter. I don’t like having spaces in my member names, so I would do the above with CrossRef(“Revenue”,”No_”) which would produce

```
"BegBalance"->"No_Scenario"->"No_Version"->"No_Entity"->"No_Product"->"Revneue";
```

If I change my syntax to CrossRef(“Revenue”,”No_”,true) I have a cross dim operator for all dimensions Except Period (uses BegBalance), and Currency, but includes year

```
"BegBalance"->"No_Year"->"No_Scenario"->"No_Version"->"No_Entity"->"No_Product"->"Revneue";
```

Used in a Fix Statement the following example

```
FIX (Aug, Actual, Working, FY15, P_000, "112")
   "111" = [[CrossRef("5800", "No_", true)]];
ENDFIX
```

would produce the following script.

```
FIX (Aug, Actual, Working, FY15, P_000, "112")
                                          "111"                =
"BegBalance"->"No_Year"->"No_Scenario"->"No_Version"->"No_Enti
ty"->"No_Product"->"5800";
ENDFIX
```

Workforce Cube Year to Date

If you use workforce, you probably have noticed the members it generates to get the month index for both the calendar and fiscal period. These can be used with this function to build a calendar to date value. The syntax is [[CYTD(memberName)]]. If you have renamed “Cal TP-Index” and “Fiscal TPIndex,”, then you have to specifically name the members in two additional parameters and the syntax is [[CYTD(memberName, calTpIndexName, fiscalTPIndexName)]]. This method is really easy to use and looks like this.

```
Fix (NOV, Actual, Working, FY15, P_000, "112")
   "5800" = [[CYTD("6100")]];
ENDFIX
```

If the default names are changed, it would look a little different.

```
Fix (Dec, Actual, Working, FY15, P_000, "112")
    "5800" = [[CYTD("6100", "Cal TP-Index", "Fiscal
TPIndex")]];
ENDFIX
```

Get ID for String

This doesn’t solve all the problems around Smart Lists and text accounts, but it is a step in the right direction. If you don’t know, both of these are held in the Planning repository. Essbase ONLY stores numbers. Look at this as the index to the value you see in Planning. The repository has the map from index to value. In an Essbase calculation, you can’t set a text account to a text value. Well, actually, you can. The syntax for this function, which assigns a text value, is [[PlanningFunctions.getIdForString("text")]]. This allows you to set the value of a text account to a string.

In Planning, you have an account named “acct1 text” that is of type text. You want to copy your values from FY16 Dec to FY17 Mar, and change the text account to “Not Budgeted,” it would look like this.

```
FIX (Actual, Working, P_000, "210")
   DATACOPY FY16->Dec TO FY17->Mar;
   Mar(
                              "acct1    text"->FY17    =
[[PlanningFunctions.getIdForString("Not Budgeted")]];
   )
ENDFIX
```

That’s A Wrap

One last thing. If you use any of these in a member formula, for some reason you have to remove a bracket on each side. So, instead of two, you just need one. I will say I have not tested all of these, but the ones I have tested/used do follow this pattern. Hopefully Oracle keeps expanding these. Although they aren’t as helpful as they were prior to Groovy, they are simpler to use than implementing a Groovy solutions for some of these needs. For you lifers, it is things like this that a newb tells you. Don’t ever think you can’t learn from somebody that “knows nothing.”