Adventures in Groovy – Part 41: RTP Interpretation Modes

Have you ever used a variable and received this error? Error: Unable to retrieve variable [variable name] deployed in the application [app name] Rule [app name].[plan type name].[rule name]. You likely saw this when a Groovy variable was used inside of {}. I finally had the issue explained to me working with the Oracle PBCS development group today. God bless them for being so gracious to help me through some of these issues! You know who you are, and I can't thank you enough for your time!

ePBCS interprets Groovy before the Groovy compiler is engaged. There are multiple interpretation modes (my words) and varies based on whether run time prompts are initiated by including /*RTPS:*/ in the calculation.

WHAT YOU NEED TO KNOW

Long story short is this, but I encourage you to read on for a deeper dive into what is happening.

EPBCS parses all Groovy scripts before executing it to identify the run time prompts used by the script. The way it identifies the run time prompts is by looking for explicit declaration of the run time prompts in a comment in the following format:

/*RTPS: {rtpName}*/

The reason for this is because there are many expressions (closures, string interpolation etc) in Groovy that use curly braces so the old way of defining run time prompts in curly braces {rtpName} is no longer recommended. Instead, the use of rtps.rtpName is recommended to reference run rime prompts inside a Groovy script.

Without the explicit declaration in the /*RTPS */ comment, the parser will try to interpret Groovy expressions inside {} as run time prompts causing the following error:

Error: Unable to retrieve variable [variable name] deployed in the application [app name] Rule [app name].[plan type name].[rule name].

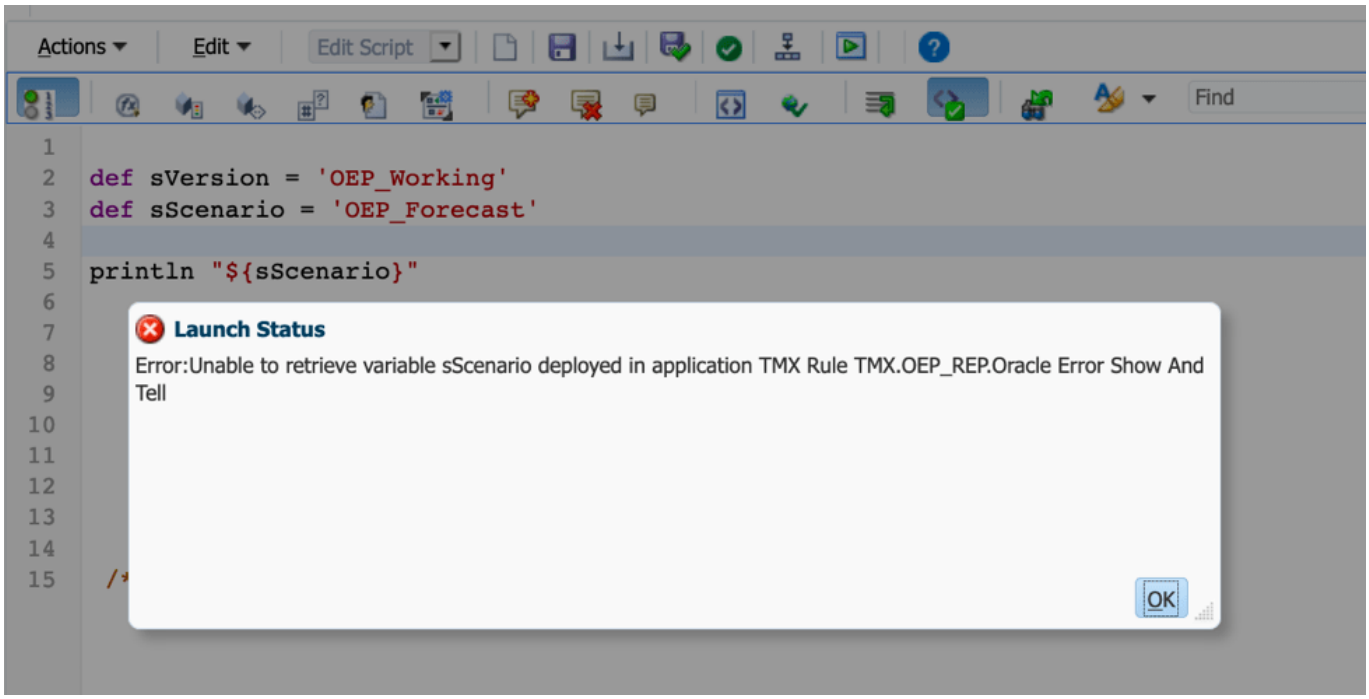
If you are using RTPs without defining them in the comment, I recommend that the script be updated to add the RTPS comment. The interpretation method of assuming everything in {} is an RTP will be deprecated in future releases. To conclude, if no RTPs are used, add /*RTPS:*/ to the script so that variables inside the {} are interpreted as Groovy variables.

Example: Interpretation Without RTPS Comment

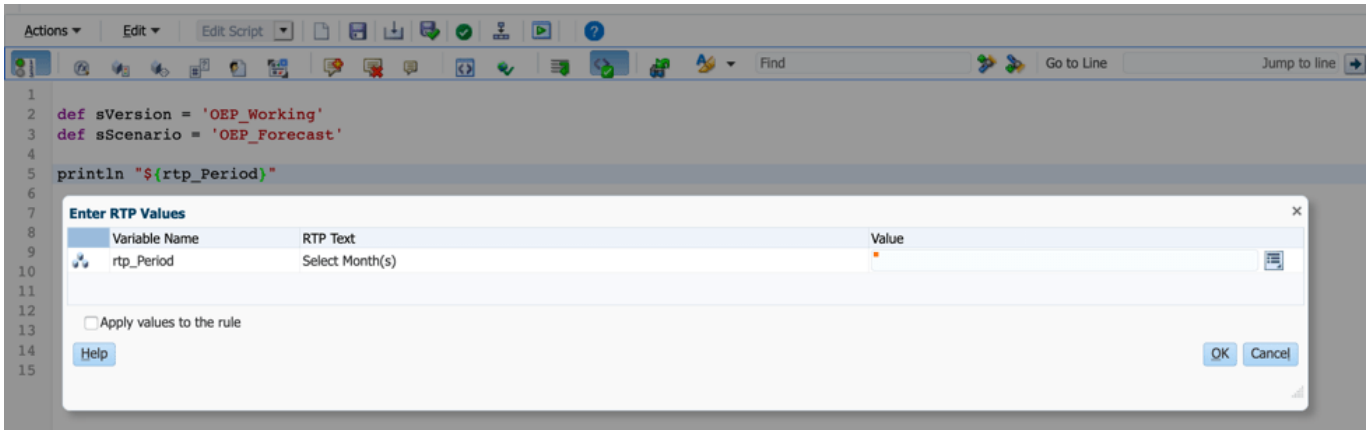
If you have worked with run time prompts, you know (or thought you knew) that they had to be defined in what looks like a comment.

/*RTPS: {rtp_Period} {rtp_Year}*/

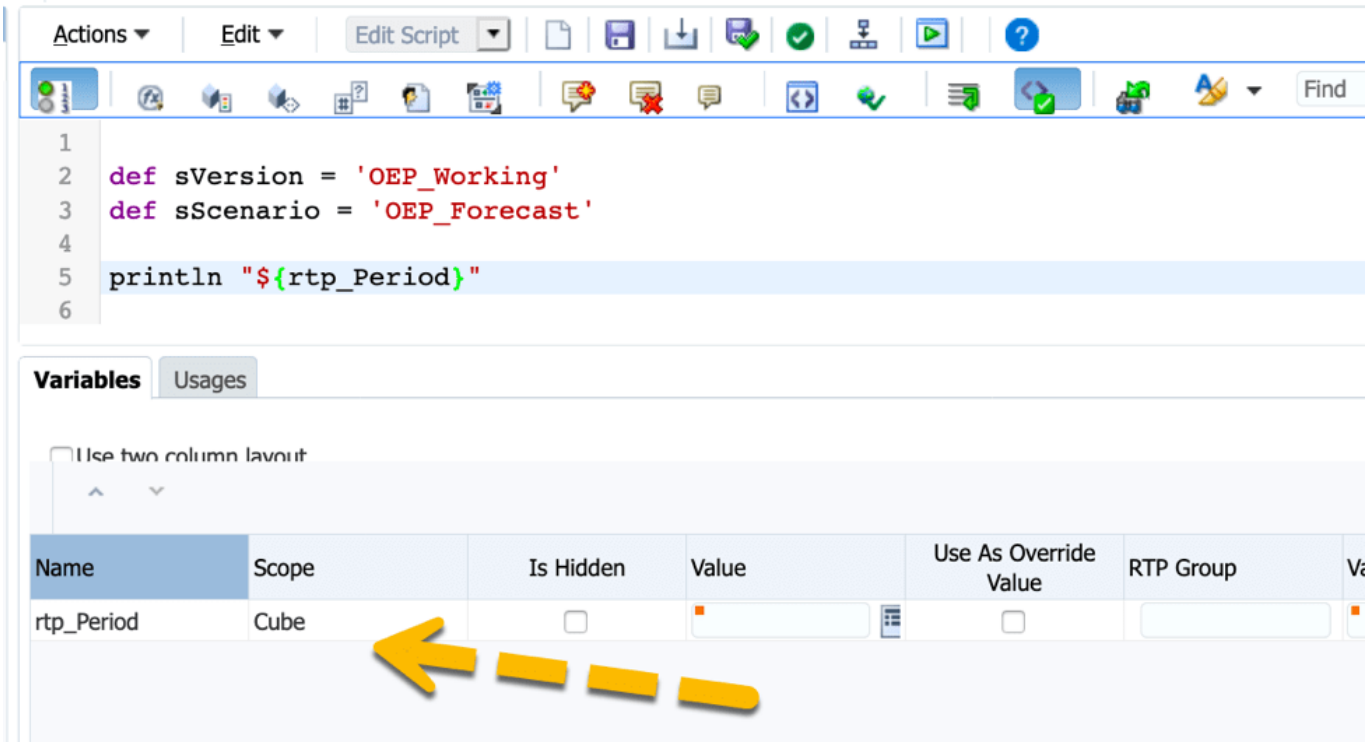
This is not true, however. Theoretically, the /*RTPS:*/ doesn’t have to be added to use run time prompts, which was news to me. When /*RTPS:*/ is EXCLUDED from a calculation, the interpreter will replace anything identified with {} as a run time prompt. If like me, I had never used a variable with the same name as a valid RTP, you would simply get an error and likely not understand why you can’t reference the variable.



If by chance you actually used a variable with the same name as an existing RTP, you might have figured this out on your own because you would be prompted for the RTP. Notice that no header was added to include RTPs in this calculation (void of any /*RTPS:*/) but still get a prompt.



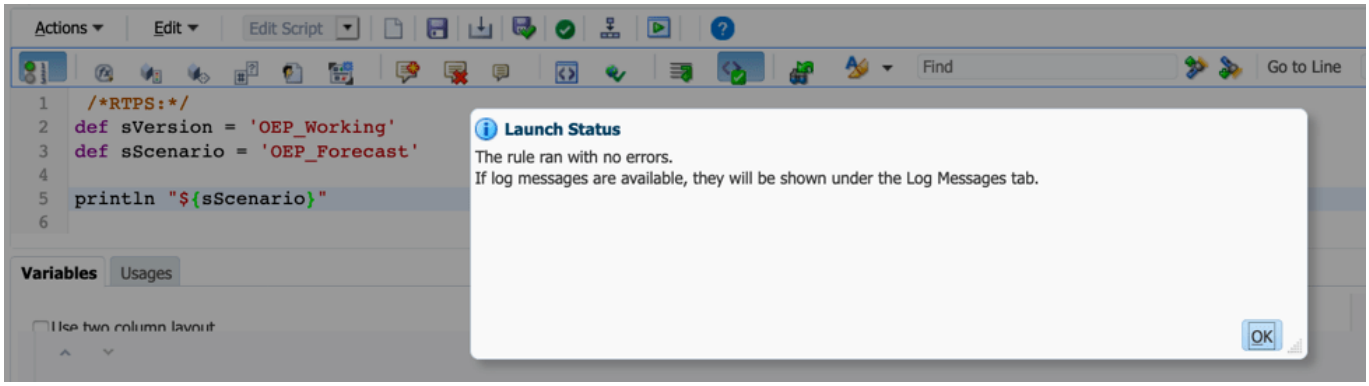
You would also notice that the RTP is identified in the Variables tab!



Now that you know this, forget that you do and don't every use RTPs without the /*RTPS:*/. This was something that was missed in initial releases and will be deprecated. So, if you don't add your RTPs in a /*RTPS:*/ and use the RTPs, this will not work in the future.

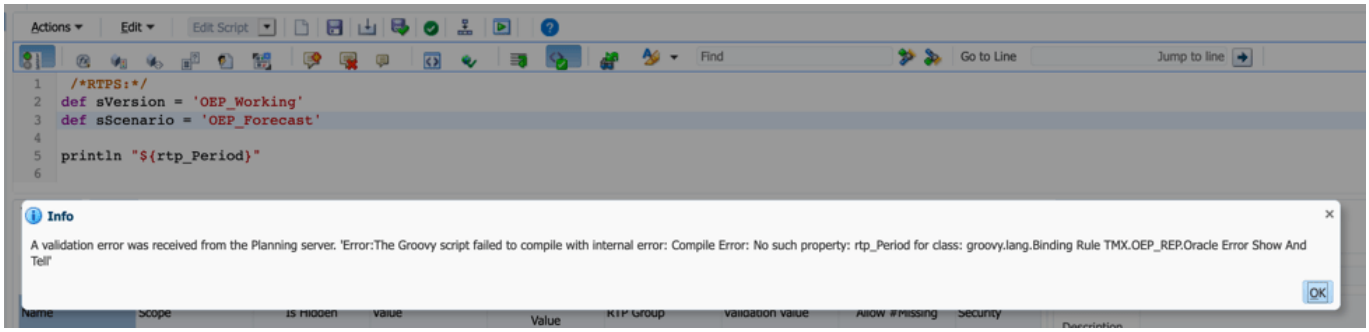
Example: Interpretation With RTPs

When /*RTPS:*/ is used anywhere in the calculation, the variables in a conventional Groovy way. In this example, sScenario is actually referencing the Groovy variable.



The same script with the RTPS inclusion header now works as expected! sScenario is no longer interpreted as an RTP. If

the variable was named the same as a valid RTP, it would also be handled as expected. Or, it is handled as I would expect it not knowing {} meant RTP without the RTPS header. rtp_Period is a valid run time prompt. Now that /*:RTPS*/ is added to the script, the same line of code is looking for a Groovy variable, not a RTP. Because the RTP is not defined, the compiler returns an error.



What I Learned Today

Talking to the dev group today was awesome because I learned a number of things.

1. There are two interpretation modes laid out above.
2. Calculations can have /*RTPS:*/ with no variables. I never really thought about doing this. Now that I understand the multiple modes, I am going to add this to every Groovy calculation to avoid any issues like this.
3. /*RTPS:*/ can be anywhere in the calculation. I don't know why it would benefit somebody to have it at the end, but it would work the same way as if it was the first line. The reason it does this is because the modules will add RTPs for certain situation and not others. For this to be possible, the need to add multiple RTPS comments with the appropriate RTPs through the calculation was required.
4. /*RTPS:*/ can exist on multiple lines. If you wanted to have each variable referenced on different lines, /*RTPS:*/ can be repeated as many times as needed.

There are a couple wins for me now that I know this.

1. I use common code and functions in scripts that I embed in groovy calculations to eliminate repetitive functionality. If these functions require an RTP (even a hidden one with an override that is a subvar), it can be referenced inside the script rather than putting them in the rules that reference the script.
2. The rule using the script can also have its own RTPs that are not needed for the common code. So, I can have common RTPs in the script and also have RTPs needed for each specific rule in that rule and not have any conflicts.

Does This Seem Irrelevant?

If you are asking yourself why this would ever come up and why any variable would be referenced inside squiggly brackets, this might help. The example above is simple and the println could have been written without the quotes and squiggly brackets.

println sScenario

You are correct, this would have worked. But, let's say you need the current month and year together. Yes, there are other ways to accomplish this, but it emphasizes the need. Let's say I need the current month concatenated with the current year formatted as FYyy.

```
def Year = (new Date()).format('yy') // produces 19
def Month  = (new Date()).format('MMM') // produces Apr
println "${Month}FY${Year}" // produces AprFY19
```

A second example would be referencing variables in a dynamic FIX statement

```
if (uniquePeriodNames.size() == 0){
  println("No cells were edited")
}
else{
    operation.grid.dataCellIterator({DataCell  cell  ->
```

```
cell.edited}).each{
    lstProduct.add(it.getMemberName("Product"))
    lstPeriod.add(it.getMemberName("Period"))
    lstYears.add(it.getMemberName("Years"))
    }

  List povmbrs = operation.grid.pov

  StringBuilder strEssCalc = StringBuilder.newInstance()
  strEssCalc <<"""
    FIX("${lstPeriod.unique().join('","')}",
        "${lstYears.unique().join('","')}",
        "${lstProducts.unique().join('","')}",
        "${povmbrs*.essbaseMbrName.join('","')}")

        "Revenue" = "REV_BASE_PRICE" * "REV_SALES_QUANTITY";
      ENDFIX
    ENDFIX
  """
}
```

Another example would be concatenating member names and country codes from an attribute dim. Suppose you want to concatenate an _USD to a country code to reference an FX rate.

$CountryCode_USD tries to reference a variable named CountryCode_USD. Using the squiggly brackets will enable this to happen without confusion. ${CountryCode}_USD would concatenate the value of CountryCode with _USD.

You can use additions and do $CountryCode + “_USD”, but it is terribly inefficient. Once is not an issue but if it is used inside of loops, it can cause performance issues.

Last Call

I don’t think it is fair to call this a bug. I do think there is a lack of clarity. The dev group is actually looking at addressing this in a future release. Again, my worlds, but changing this to always referencing things inside of brackets

as a Groovy variable would make it a bit less confusing. Basically, nothing inside of {} would be interpreted as an RTP unless it is defined in the header definition. So, something in brackets that is defined as an RTP would be an RTP. Anything else would be assumed a variable.