

# Adventures in Groovy – Part 7: Validating Run Time Prompts

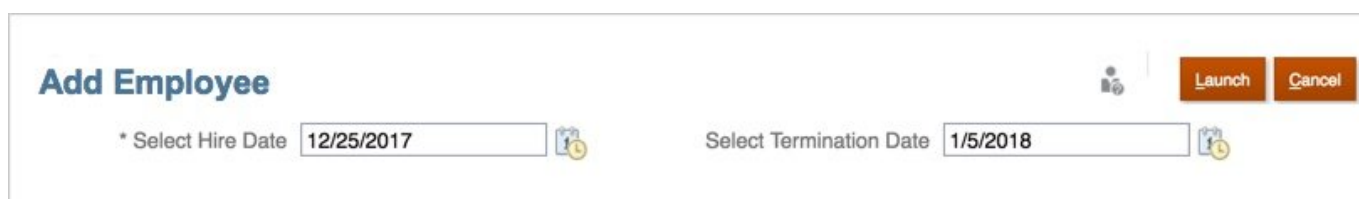
## Introduction

When developing applications in PBCS, there are times when we need to request information from users. This is especially useful in workforce application, but certainly not limited to them. With Groovy, we can take validation to the next level. We can compare inputs to each other, we can limit input, and we can now stop the execution of the calculation until the inputs are validated. The difference now is that we have the ability to keep the prompts open, and force the user to either enter valid data, or cancel the request.

## Validating Hire Date / Termination Date Use Case

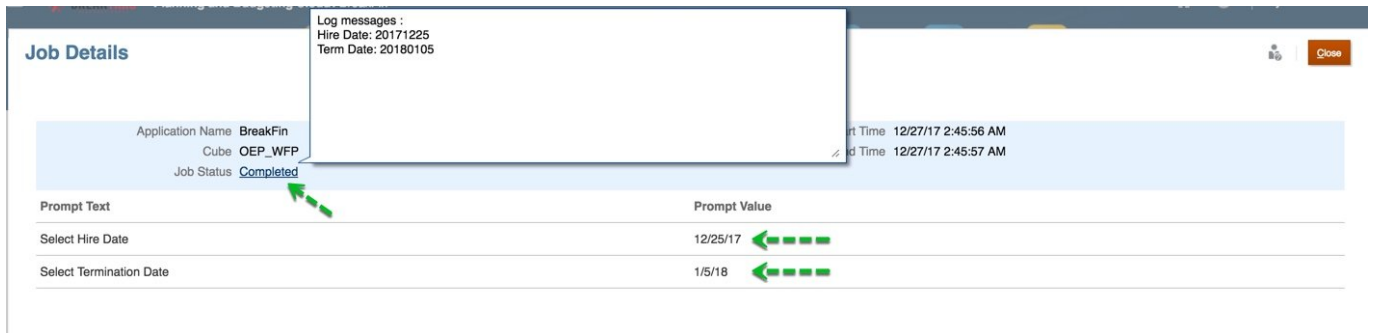
Let's start with a very simple example. Let us assume we are adding a new employee, and we are asking for a hire and termination date. A real-life example would require more options, like a name, title, union, and other required inputs. To simplify this tutorial, we are only entering the hire and termination dates to prove out the validation and show functionality of the Groovy RTP validation logic.

When a user enters a termination date after a hire date and launches the rule, it validates and executes the rule.



The screenshot shows a web form titled "Add Employee". It contains two date input fields: "Select Hire Date" with the value "12/25/2017" and "Select Termination Date" with the value "1/5/2018". Both fields have a calendar icon to their right. To the right of the form are two buttons: "Launch" and "Cancel".

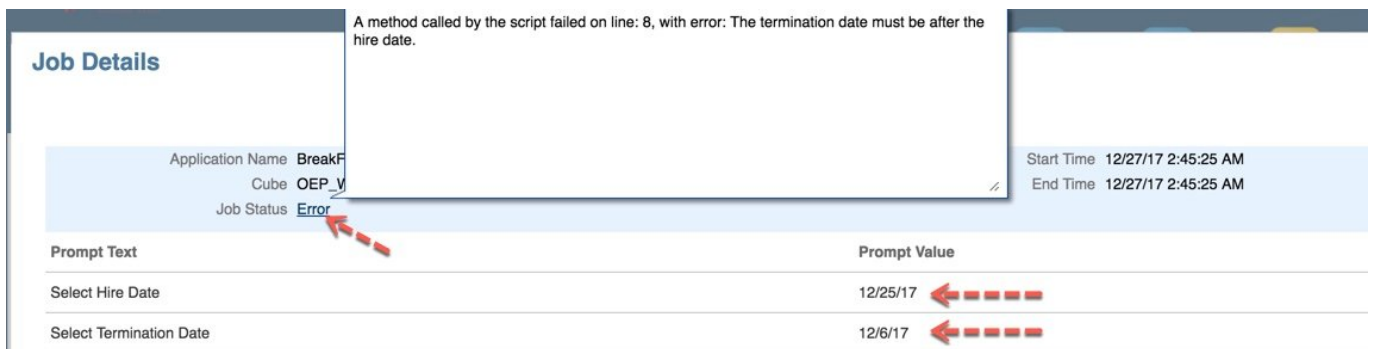
When the job is opened in the job console, we see the RTPs and the values entered, and the Job Status is selected, the log shows the values returned to Essbase.



When a user enters a termination date prior to a hire date and launches the rule, it an error is returned and the RTP window doesn't go away. At this point, the user has to correct the error, or cancel the execution of the business rule.



In this case, the log shows the business rule failed.



## Code

There are multiple objects that are used to accomplish RTP Validation. The code that processed the above action is the following.

```

/*RTPS: {RTP_HireDate} {RTP_TermDate}*/
def mbUs = messageBundle(["validation.InvalidDate":"The
termination date must be after the hire date."])
def mbl = messageBundleLoader(["en" : mbUs])

// Validate the Rtp values
if(rtps.RTP_HireDate.getEssbaseValue() >
rtps.RTP_TermDate.getEssbaseValue())
    throwVetoException(mbl, "validation.InvalidDate",
rtps.RTP_HireDate)

// Print the results to the log
println "Hire Date: " + rtps.RTP_HireDate.getEssbaseValue()
println "Term Date: " + rtps.RTP_TermDate.getEssbaseValue()

```

## **rtps object**

Creating RTPs in Groovy was covered in the previous article. If you haven't read that, it would be a good time to take a look, as it explains the basic of this object. Expanding on the use of the object, we are using some additional methods. This object has many, including returning the input as boolean, double, strings, dates, members, and smart lists, to name a few. In this example, we are using `getEssbaseValue`, which returns the value sent to Essbase and stored. If there was a need to compare date ranges, we could have used the `getDate`, and expanded on this with the Groovy date functions to get the days, months, or years between the entered values. In this simple example, we just want to make sure the term date is greater than the hire date.

## **messageBundle**

The first thing that is required is to create a `messageBundle` and `messageBundleLoader`. These two objects work together to hold the errors, the error messages, and multiple languages, if required.

The `messageBundle` is a map that holds all the errors (name and description). In this example, we only have one error, but

more can be added and separated by commas. The `messageBundleLoader` holds all the `messageBundle` objects that represent the different languages.

## **throwVetoException**

When an exception is found, executing this method will initiate an error and cause the RTP validations to fail. This method requires the `messageBundleLoader`, and the error to be returned to the user.

## **Other Use Cases**

By now you are probably already thinking of other uses of this. I can see limiting growth rates, confirming combinations of RTPs (like not allowing salaried people in a union), ensuring that a new employee doesn't have a hire date prior to the current date, and probably hundreds of other ways to use this.

If you would like to share an idea, please post a comment!

## **Conclusion**

Being able to validate user input prior to executing a calculation and returning the input request to the user is huge step forward, and just another benefit of Groovy calculations. We can reduce the number of user errors and improve the user experience.