Adventures in Groovy – Part 15: Returning Errors (RTP Edition)

Introduction

One of the huge benefits that available in Groovy Calculations is the ability to interact with a user, validate data, and act on the validation. Now, we can interrupt form saves, stop Run Time Prompts from continuing, and communicate information back to the user.

This may sound repetitive if you have read part 13 and part 14, and you can skip to the code example to learn more about run time prompt validation. If not, you must have an understanding of the validation functions and the components of the messageBundle. There are a number of functions for validation, and they can be categorized functionally. Although they all can be use somewhat interchangeably, the logical uses are

- Data Form validation functions
  - addValidationError
- RTP validation functions
  - validateRtp
- Validation functions that are more open ended and can be used just about anywhere
  - messageBundle
  - messageBundleLoader
  - throwVetoException

In this post, we will discuss one aspect of this, and probably the simplest application, validating Run Time Prompts (RTP).

The MessageBundle

Before a few of the methods can be used, one must first understand the MessageBundle and MessageBundleLoader methods. To look at documentation, they might seem very complex, and a maybe a little intimidating. The reality is that these are just objects that hold the error messages. That is pretty much the long of short of it. The messageBundle holds a map (basically a lookup table that is two columns and *n* rows) of the error ID and the description of the error you want to display. If the application is consumed by users with multiple languages, a messageBundle can be created for each language. The messageBundleLoader allows you to identify which bundle to use based on the user's local. The example below should answer any questions you have.

The Message Bundle

Think of this method as an array, or a table in Excel. It has 2 columns (ID and message). It can have an infinite amount of rows. The syntax of this is "[id,message]". For multiple errors, this is duplicated, separated by a comma, like "[id,message],[id,message]". Here is an example of a messageBundle with one error.

```
def mbUs = messageBundle( ["validation.InvalidCharacters":"Only alphanumeric characters can be entered (a-z, 1-9)."] )
```

And with two errors.

```
def mbUs = messageBundle( ["validation.InvalidCharacters":"Only alphanumeric characters can be entered (a-z, 1-9)."],
["validation.Negative":"A positive number is required."])
```

And with two errors in Spanish.

```
def mbSpanish = messageBundle( ["validation.InvalidCharacters":"Sólo se pueden introducir caracteres alfanuméricos (a-z, 1-9)."],
```

["validation.Negative":"Se requiere un número positivo."])

This can be extended to hold all the error messages required for the scope of the calculation.

The Message Bundle Loader

The messageBundleLoader is the piece that pulls either a single, or multiple, messageBundles together to use in a call. If only one language is required, it would look like this.

[def mbl = messageBundleLoader(["en" : mbUs])

For multiple languages, it would include multiple messageBundles

[def mbl = messageBundleLoader(["en" : mbUs],["en" : mbSpanish])

Validate The Input

When a validation error exists, the prompt window will not close, so it won't let a user continue unless all the data entered validates. Validations are only limited to your knowledge of how to validate the input. Let Google be your friend. You will be hard pressed to have a sitiation where you can't find an example of what you are trying to do. If you aren't familiar with "regex," it will likely be included in just about any Google search you do. The examples below all use a regex string to validate the inputs.

To use a run time prompt in Groovy, they must be initiated. This looks like a comment, but it acts differently when prefaced by RTPS:

/*RTPS: {EmployeeName} {EmployeePhone} {EmployeeEmail} */

Next, we will create a messageBundle. Although it is simplier than above, it is more than enough to demonstrate its use in the validateRtp method. This creates an error for each of the

three validations in English.

```
def mbUs = messageBundle(["validation.invalidemail":"Email address is invalid: {0}", "validation.invalidphone":"Phone number is invalid: {0}", "validation.invalidnamelength":"Employee name must be 5 to 40 characters: {0}"])
def mbl = messageBundleLoader(["en" : mbUs])
```

Now, the actionable stuff. The next 3 lines will validate the 3 run time prompts. If any of them fail, the RTP window will remain open and the user can't continue until they fix the errors or cancel the action.

```
// Validate the Rtp values
validateRtp(rtps.EmployeeName, {(5..40).contains(it.enteredValue.size())  }, mbl, "validation.invalidnamelength", rtps.EmployeeName)
validateRtp(rtps.EmployeeEmail, /^.+@.+/, mbl, "validation.invalidemail", rtps.EmployeeEmail.enteredValue)
validateRtp(rtps.EmployeePhone, /^(?:\+?1[-]?)?\(?([0-9]{3})\)?[- ]?([0-9]{3})[- ]?([0-9]{4})$/, mbl, "validation.invalidphone", rtps.EmployeePhone)
```

Putting it all together, we have the following.

```
/*RTPS: {EmployeeName} {EmployeePhone} {EmployeeEmail} {Scenario} {Year} {Period} {Entity} {Version}*/
def mbUs = messageBundle(["validation.invalidemail":"Email address is invalid: {0}", "validation.invalidphone":"Phone number is invalid: {0}", "validation.memberexists":"The member you have specified already exists and cannot be created: {0}.", "validation.invalidnamelength":"Employee name must be 5 to 40 characters: {0}"])
def mbl = messageBundleLoader(["en" : mbUs])

// Validate the Rtp values
validateRtp(rtps.EmployeeName, {(5..40).contains(it.enteredValue.size())  }, mbl, "validation.invalidnamelength", rtps.EmployeeName)
validateRtp(rtps.EmployeeEmail, /^.+@.+/, mbl,
```

```
"validation.invalidemail", rtps.EmployeeEmail.enteredValue)
validateRtp(rtps.EmployeePhone, /^(?:\+?1[-]?)?\(?([0-9]{3})\)?[- ]?([0-9]{3})[- ]?([0-9]{4})$/, mbl, "validation.invalidphone", rtps.EmployeePhone)
```

Wrap Up

It has been a long time since developers have had this kind of control. The possibilities are only limited by your imagination and business requirements, but there isn't any validation that can't be done. This wraps up the 3 validation methods.

Enjoy this new functionality. Don't underestimate its importance. This functionality can save your customers hours of work and lots of frustration. Helping them input accurate data improves the forecasting and budgeting process. Implement these techniques and they will love you!