Adventures in Groovy – Part 13: Returning Errors (Data Forms)

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. 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, the id:message 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 in all the locales required.

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, or multiple messageBundles, they would be concatenated together with commas. View a valid list of locales to make sure the parameter in parenthesis is correctly linked to the correct locale.

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

Throw an Exception (Interrupt Form Save)

Here is where the cool stuff happens. see post about looping through cells

If a validation error exists, an exception can be generated to stop the form from saving. To do this, simply use the throwVetoException method. This accepts 2 parameters. The first is the messageBundlerLoader, and the second is the id associated to the to be displayed. Using the example above, and assuming the local is US, the following would stop the form from saving and display a message of "Only alphanumeric characters can be entered (a-z, 1-9)."

```
throwVetoException(mbl, "validation.InvalidCharacters")
```

Consolidated Example

The following example creates two error messages in two languages. On form save, this will loop through all the cells and throw an error if any value is negative.

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

def mbSpanish = messageBundle(
["validation.InvalidCharacters":"Sólo se pueden introducir
caracteres alfanuméricos (a-z, 1-9).",
"validation.Negative":"Se requiere un número positivo."])

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

operation.grid.dataCellIterator.each {
  if(it.data < 0)
    throwVetoException(mbl, "validation.Negative")
  }
```

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. Future posts will tackle validating Run Time Prompts, and taking form validation one step further by adding cell level tool-tips and color coding.

The last thing with these validation calculations is the importance of when they are executed. The documentation I have from Oracle states something slightly different, so I don't know if this is the way it is supposed to work, but in my experience, where the rule runs is critical. Here is what I am experiencing.

- When the rule is set to Run Before Save, and there is a validation error, the user can't save the form and an error messages is displayed in the correct locale. To me, this is the experience that is expected.
- When the rule is set to Run After Save (which is the way it is documented), and there is a validation error, the user receives an error, but the data is saved.

The difference in the above does provide some interesting options. Let's say that we have a form and users are required to allocate an expense. If the expense is not allocated at 100%, the form can't be saved. Assume that there is a rule that the expense shouldn't be allocated to more than 3 places, but users should be warned if it is. In this case, if the rule is set to run AFTER save, the user gets the message, but the data is saved.

Either way, if the rule is executed before other rules on the form, no subsequent form will fire if there is a validation error.