Adventures in Groovy – Part 2: Data Validation

Introduction

We all know the Data Form validation rules are serviceable, but they are not robust. When Smart View advanced and forms were opened in Excel, the validation logic developers had in JavaScript became useless. Since then, we have really missed the ability to communicate with the user interactively with visual cues and validation rules that halted the saving of data. Well, Groovy calculations to the rescue!

I will preface with the fact that I am encountering some odd behavior, so I am going to break this up into multiple articles. It appears that Oracle is validating Groovy enhancements in Data Forms on the web, and not necessarily testing the full functionality in Smart View. Currently, I have this working in a browser perfectly, but 3 of the 8 columns are failing in Smart View. I am hoping to get closure to a ticket on this in the near future. When I get a resolution, I will amend this article with some clarity on either what I am doing wrong, or when it will be resolved.

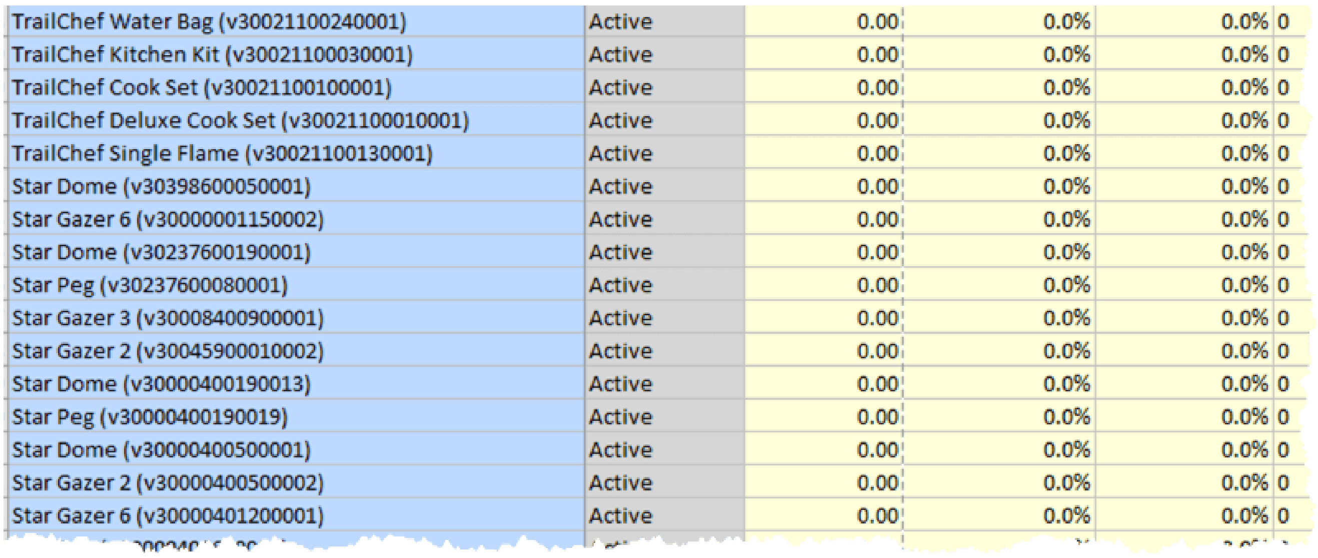
High Level requirement

At a high level, the planners want to see any seeded value that was changed with a different background color to single out the lines that have been edited.

The Details

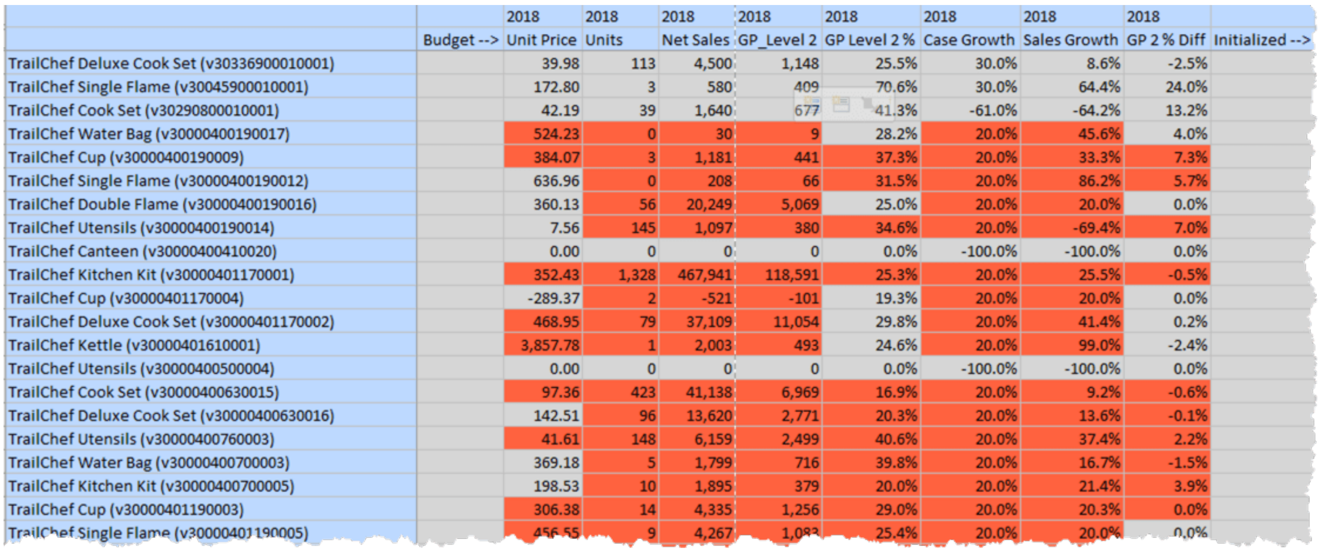
We have a form that provides the users the ability to override seeded data. In this example, a planner can change the Average Price/Case, Net Sales, and/or GP Level 2 at any level of the hierarchy and gets allocated down to level 0 on a % to

Total. This form has the accounts in question for 3 sources. The override columns are a separate version that is set to top down so security doesn’t prevent them from entering at a non-level 0 member. This is only used to enter the 3 values, is used to calculate the Input source, and is cleared.



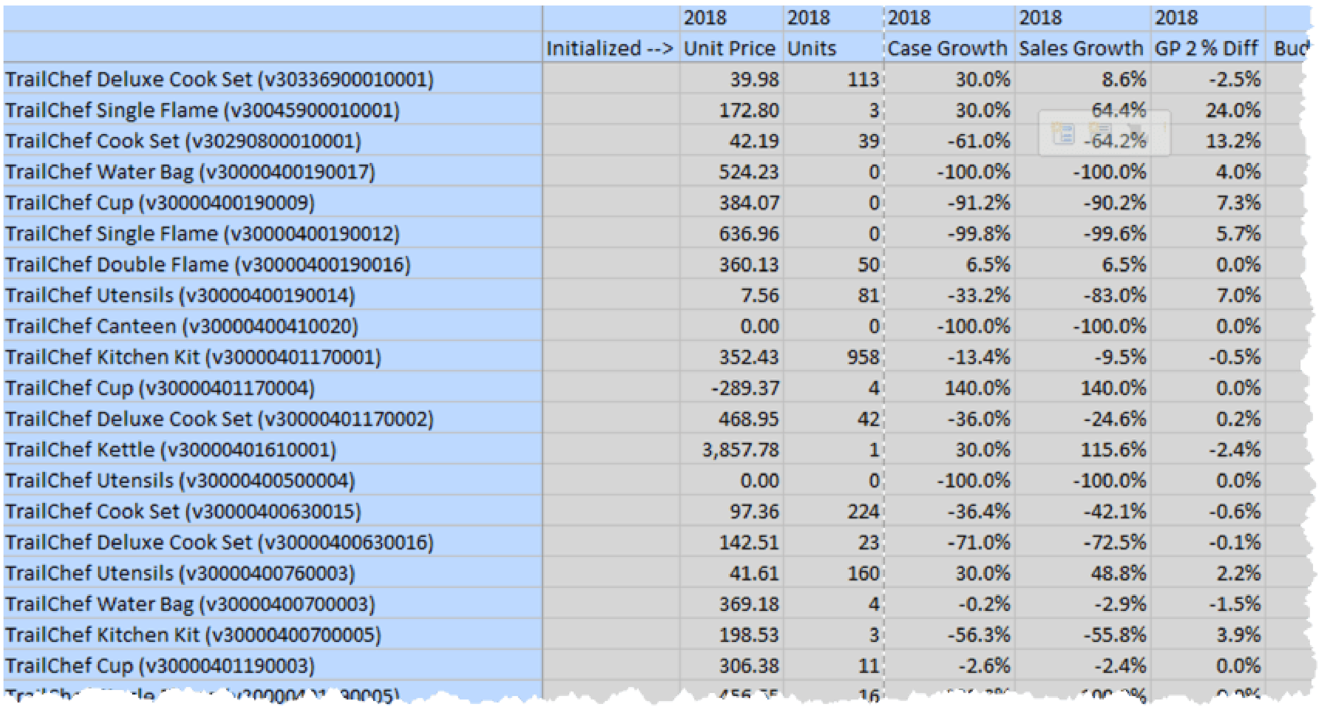
TrailChef Water Bag (v30021100240001)	Active	0.00	0.0%	0.0%	0
TrailChef Kitchen Kit (v30021100030001)	Active	0.00	0.0%	0.0%	0
TrailChef Cook Set (v30021100100001)	Active	0.00	0.0%	0.0%	0
TrailChef Deluxe Cook Set (v30021100010001)	Active	0.00	0.0%	0.0%	0
TrailChef Single Flame (v30021100130001)	Active	0.00	0.0%	0.0%	0
Star Dome (v30398600050001)	Active	0.00	0.0%	0.0%	0
Star Gazer 6 (v30000001150002)	Active	0.00	0.0%	0.0%	0
Star Dome (v30237600190001)	Active	0.00	0.0%	0.0%	0
Star Peg (v30237600080001)	Active	0.00	0.0%	0.0%	0
Star Gazer 3 (v30008400900001)	Active	0.00	0.0%	0.0%	0
Star Gazer 2 (v30045900010002)	Active	0.00	0.0%	0.0%	0
Star Dome (v30000400190013)	Active	0.00	0.0%	0.0%	0
Star Peg (v30000400190019)	Active	0.00	0.0%	0.0%	0
Star Dome (v30000400500001)	Active	0.00	0.0%	0.0%	0
Star Gazer 2 (v30000400500002)	Active	0.00	0.0%	0.0%	0
Star Gazer 6 (v30000401200001)	Active	0.00	0.0%	0.0%	0

The Initialized source is seeded from prior year growth. This, in essence, is the basline seeded amount. At initialization, the Input source is a duplicate of Initialized source.



		2018	2018	2018	2018	2018	2018	2018	2018	
	Budget -->	Unit Price	Units	Net Sales	GP_Level 2	GP Level 2 %	Case Growth	Sales Growth	GP 2 % Diff	Initialized -->
TrailChef Deluxe Cook Set (v30336900010001)		39.98	113	4,500	1,148	25.5%	30.0%	8.6%	-2.5%	
TrailChef Single Flame (v30045900010001)		172.80	3	580	409	70.6%	30.0%	64.4%	24.0%	
TrailChef Cook Set (v30290800010001)		42.19	39	1,640	677	41.3%	-61.0%	-64.2%	13.2%	
TrailChef Water Bag (v30000400190017)		524.23	0	30	9	28.2%	20.0%	45.6%	4.0%	
TrailChef Cup (v30000400190009)		384.07	3	1,181	441	37.3%	20.0%	33.3%	7.3%	
TrailChef Single Flame (v30000400190012)		636.96	0	208	66	31.5%	20.0%	86.2%	5.7%	
TrailChef Double Flame (v30000400190016)		360.13	56	20,249	5,069	25.0%	20.0%	20.0%	0.0%	
TrailChef Utensils (v30000400190014)		7.56	145	1,097	380	34.6%	20.0%	-69.4%	7.0%	
TrailChef Canteen (v30000400410020)		0.00	0	0	0	0.0%	-100.0%	-100.0%	0.0%	
TrailChef Kitchen Kit (v30000401170001)		352.43	1,328	467,941	118,591	25.3%	20.0%	25.5%	-0.5%	
TrailChef Cup (v30000401170004)		-289.37	2	-521	-101	19.3%	20.0%	20.0%	0.0%	
TrailChef Deluxe Cook Set (v30000401170002)		468.95	79	37,109	11,054	29.8%	20.0%	41.4%	0.2%	
TrailChef Kettle (v30000401610001)		3,857.78	1	2,003	493	24.6%	20.0%	99.0%	-2.4%	
TrailChef Utensils (v30000400500004)		0.00	0	0	0	0.0%	-100.0%	-100.0%	0.0%	
TrailChef Cook Set (v30000400630015)		97.36	423	41,138	6,969	16.9%	20.0%	9.2%	-0.6%	
TrailChef Deluxe Cook Set (v30000400630016)		142.51	96	13,620	2,771	20.3%	20.0%	13.6%	-0.1%	
TrailChef Utensils (v30000400760003)		41.61	148	6,159	2,499	40.6%	20.0%	37.4%	2.2%	
TrailChef Water Bag (v30000400700003)		369.18	5	1,799	716	39.8%	20.0%	16.7%	-1.5%	
TrailChef Kitchen Kit (v30000400700005)		198.53	10	1,895	379	20.0%	20.0%	21.4%	3.9%	
TrailChef Cup (v30000401190003)		306.38	14	4,335	1,256	29.0%	20.0%	20.3%	0.0%	
TrailChef Single Flame (v30000401190005)		456.55	9	4,267	1,083	25.4%	20.0%	20.0%	0.0%	

The Initialized source is also on the form. When overrides are entered, it is applied to the input source. At this point in the process, the Input is different from the Initialized source, as shown by the orange color in the previous image.



		2018	2018	2018	2018	2018	
	Initialized -->	Unit Price	Units	Case Growth	Sales Growth	GP 2 % Diff	Bud
TrailChef Deluxe Cook Set (v30336900010001)		39.98	113	30.0%	8.6%	-2.5%	
TrailChef Single Flame (v30045900010001)		172.80	3	30.0%	64.4%	24.0%	
TrailChef Cook Set (v30290800010001)		42.19	39	-61.0%	-64.2%	13.2%	
TrailChef Water Bag (v30000400190017)		524.23	0	-100.0%	-100.0%	4.0%	
TrailChef Cup (v30000400190009)		384.07	0	-91.2%	-90.2%	7.3%	
TrailChef Single Flame (v30000400190012)		636.96	0	-99.8%	-99.6%	5.7%	
TrailChef Double Flame (v30000400190016)		360.13	50	6.5%	6.5%	0.0%	
TrailChef Utensils (v30000400190014)		7.56	81	-33.2%	-83.0%	7.0%	
TrailChef Canteen (v30000400410020)		0.00	0	-100.0%	-100.0%	0.0%	
TrailChef Kitchen Kit (v30000401170001)		352.43	958	-13.4%	-9.5%	-0.5%	
TrailChef Cup (v30000401170004)		-289.37	4	140.0%	140.0%	0.0%	
TrailChef Deluxe Cook Set (v30000401170002)		468.95	42	-36.0%	-24.6%	0.2%	
TrailChef Kettle (v30000401610001)		3,857.78	1	30.0%	115.6%	-2.4%	
TrailChef Utensils (v30000400500004)		0.00	0	-100.0%	-100.0%	0.0%	
TrailChef Cook Set (v30000400630015)		97.36	224	-36.4%	-42.1%	-0.6%	
TrailChef Deluxe Cook Set (v30000400630016)		142.51	23	-71.0%	-72.5%	-0.1%	
TrailChef Utensils (v30000400760003)		41.61	160	30.0%	48.8%	2.2%	
TrailChef Water Bag (v30000400700003)		369.18	4	-0.2%	-2.9%	-1.5%	
TrailChef Kitchen Kit (v30000400700005)		198.53	3	-56.3%	-55.8%	3.9%	
TrailChef Cup (v30000401190003)		306.38	11	-2.6%	-2.4%	0.0%	
[illegible]		[illegible]	16	[illegible]	[illegible]	[illegible]	

Why Not Validation Rules?

First, there is limited functionality in the Data Form validation rules. In this case, the functionality is there, but has an issue with the precision of the data. Even though Input equals Initialized (or appears to), validation fails and shows a different background color. I have seen this before with decimals with large precision.

How Groovy Solves This

Groovy calculations have the ability to traverse through the cells of a Data Form. The 8 cells that can be impacted by the 3 overrides can be checked against their counterpart in subsequent columns (comparing the same account in the Input source to the Initialized source). This is for another discussion, but Groovy can actually create temp grids and pull data directly from Essbase that doesn't exist in the grid, too.

To simplify this, the following only loops through the first column – Avg Price / Case. This can be replicated easily for all subsequent columns by changing the account in question.

This example uses several Groovy methods/functions. First, the data grid is stored in a variable, as it will be referenced throughout. Next, we are using the dataCellIterator, which is the same in the previous post on Groovy. If you didn't read that, or don't understand the iterator, check that out.

At this point, the calculation is requesting to loop through all the cells with Avg Price/Case AND Input in the POV. Inside the loop, lDestMembers is set to a list equal to all the members in the POV for the relative cell. memberNames returns every member in the POV in a Groovy list.

The next step is getting the value for the corresponding cell in the Initialized source. getCellWithMembers accomplishes this with the appropriate parameters passed. This function accepts member names, so all the members in the Input cell's POV are used, excluding the source dimension. This is changed to Initialize.

Lastly, the comparison is made between the two cells. If they are not identical, setBgColor is executed on the Input source cell to identify it as something that has changed due to an override.

The Calculation

```
// Initialize a grid
DataGrid curDataGrid = operation.grid
// Set the color to be used if the values are not identical
def iColor = 16746496
// Loop through the cells in column that has
//  Average Price/Case and Input in the POV
operation.grid.dataCellIterator('Avg_Price/Case','Input').each
  {
    // Get the POV for the cell
    def lDestMembers = it.memberNames
     // get the value in the Initialized source that is
equivalent to
    // the cell in the Input Source.  The POV form the Input
source
    // is used with the exception of the source is changed to
Initialize
                                  def        dValue        =
operation.grid.getCellWithMembers(lDestMembers[0].toString(),
    lDestMembers[1].toString(),lDestMembers[2].toString(),
    lDestMembers.toString(),lDestMembers[4].toString(),

lDestMembers[5].toString(),"Initialize",lDestMembers[7].toStri
ng(),
    lDestMembers[8].toString(),lDestMembers[9].toString(),
    lDestMembers[10].toString()).data
     // if the value is different between the Input and
Initialized source,
    // change the background color
    if(it.data != dValue)
    {
      it.setBgColor(iColor)
    }
  }
```

Data Form Changes

This new Groovy Business Rule should be added to the form and executed on load and save. This will ensure that the accounts that have been changed are identified both before, and after, the user makes any changes. One more note that might save you hours of frustration – make sure this rule runs last when other rules are also executed!

Conclusion

This opens up a lot of options that far surpass the default form validations. Other options are available.

- Tool-tips can also be assigned to a cell instructing the user how to resolve a validation error, if one exists.

- The form save can be interrupted, stopping the user from saving data on a form (or even saving only parts of the form) when validation errors exist.
- Data can be altered to force validation prior to saving.
- Detailed messages can be displayed with instructions and other communication to the user.
- Have specific calculations executed based on the data entered.

This is not an exhaustive list. We, as developers and architects, literally can do anything we want and have complete control over what happens and what doesn’t happen. This is exciting because we have nearly complete control over what happens on save. If you have other ideas, or questions, please share them with comments.