

Formula Editor Deep Dive

JMP Discovery Summit: March 12-14, 2019

**Presenters/Authors**

Mike Muhlada

JMP Senior Development Tester

Audrey Shull

JMP Senior Testing Manager

**Abstract**

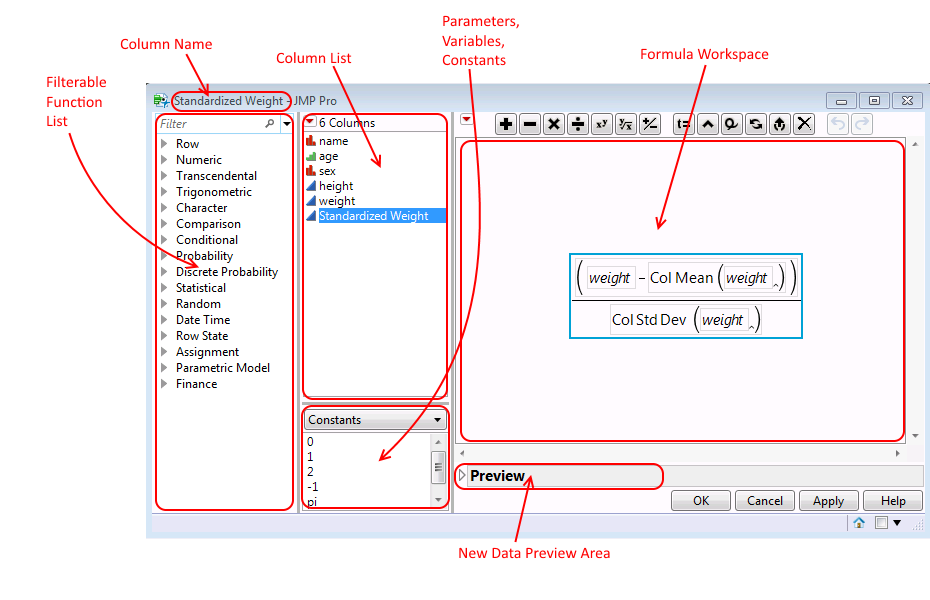
This demo will use examples to explain Formula Editor mechanics, to investigate and fix common errors, and to solidify best practices and useful ‘tricks’ for formula creation. A detailed walkthrough will cover various aspects of the formula builder, including using drag and drop, keyboard shortcuts, and function buttons to create formulas. As we work the examples, we will discuss useful functions, uncover and correct common mistakes, work with subscripts, and more. Best practice topics to be covered include: custom formats, using formulas for data ‘cleanup’, column transforms and using intermediate columns to build more complex formulas. Special emphasis will be placed on JMP 14 new features including User-defined functions, result preview, ‘live’ formula error/warning detection, and customizable function list.

**Introduction**

JMP column formulas enable users to compute values for each row in a column. Formulas are stored as part of that column’s information, along with other column properties like Column Name, Data Type, Modeling Type, etc. These formulas are created using JMP’s Formula Editor window, recently overhauled in JMP version 13 and enhanced in version 14.

See the screenshot below for an overview of different areas of the Formula Editor.

* Far left pane displays the functions list in an expandable tree. This list is filterable.
* Next we have the selectable Column variables list.
* Below that, the Parameters, Local and Table Variables and Constants pane.
* The majority of the area is occupied by the workspace. This area has calculator style function buttons across the top, with a large working area to create and display this column’s formula.
* Expandable Preview area (NEW) shows subset of what column values will be using this formula.
* The formula editor window retains the last size you used. This is nice if you are usually working with larger formulas.



**Screenshot: JMP 14 Formula Editor**

**Column Formula overview**

Column formulas operate on each Row of the data table. There is an implied ‘for each row()’. This works well when your formulas are based upon data in different columns:

e.g. :Col 1 + log (:Col 2).

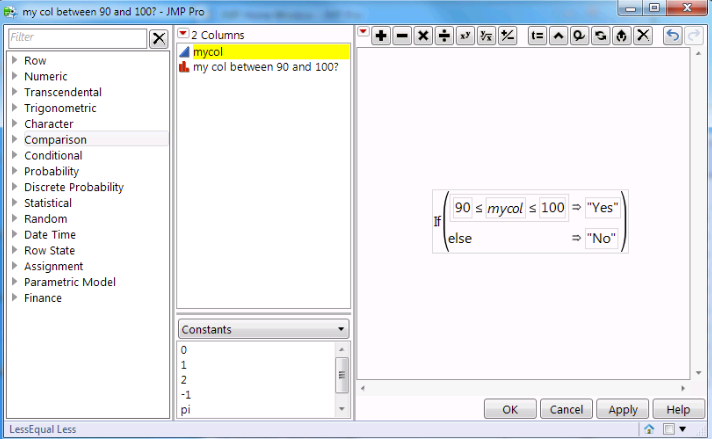
**Example from the JMP community**

<https://community.jmp.com/t5/Discussions/Need-help-with-the-formula/m-p/37028> .

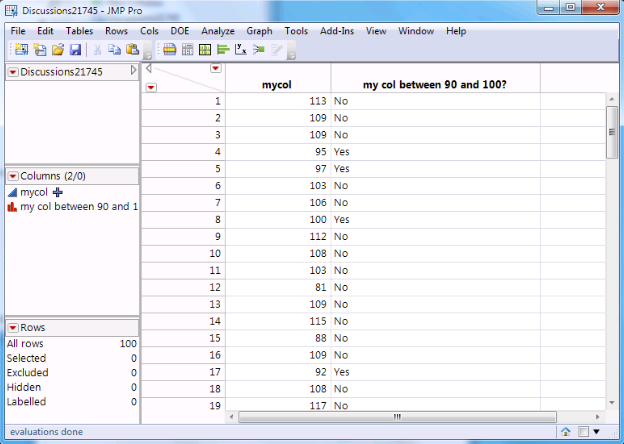
Re-stated problem statement:

*User wanted to generate a new column formula that tests if values in given column (mycol) are within a range of 90-100. If so, display ‘Yes’, otherwise ‘No’.*

This solution from Jeff Perkinson is simple and clean. It illustrates that the formula is applied to each row independently. Here are screenshots of the formula and the resulting data table.



**Screenshot: Simple formula testing if each row’s value between 90-100**



**Screenshot: Resulting data table with new column**

The previous example only required data from the current row for its computations. For the cases where we need to compute a value for ACROSS rows, e.g. the Mean of all values in this column, there are column versions of functions as well. Col Mean(:height).

More complex formulas might require saving or accessing data from previous rows. Since column formulas are really JSL scripts, they are flexible and powerful.

More detailed explanations of the basic Formula Editor concepts, as well as a few more tips and tricks, are available in the **Appendix** at the end of this document.

**JMP DEMO**

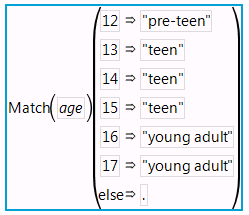
The rest of this document follows the JMP Formula Editor tutorial demo. We hope you can use this area to take notes.

Goals for ‘*Height/Age’*:

* Type formula directly into Tiny Editor
* Column variables can be typed using leading ‘:’ or not
* Double-click to replace/fix. Drag/drop is even easier
* Use Undo/Redo
* Edit existing formula by clicking on ‘+’ next to column name in column list

Goals for ‘*Match()*’ formula:

* Show formula tree with flyout menus or tree expansion
* Automatically pre-populate ‘*Match()*’ function using Conditional Flyout menu
* ‘X’ clears the formula
* Move around workspace using arrow keys
* Editing with Tiny and Full-screen JSL script editors
* Copy/paste using context clicks, Ctrl-C/Ctrl-V, or drag/drop
* Shift-click on *Match()* will auto-populate.



Goals for *Col Mean()* and ‘byVariable’ formula:

* Explain column versions of formulas
* Searchable Function list
* Add ‘byVariable’ using commas or the insert button.
* Round *Col Mean()* result directly:



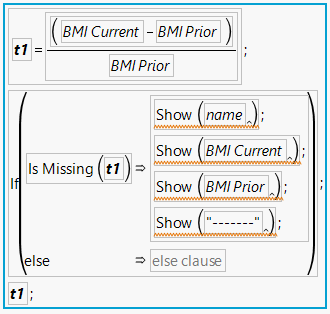
* OR with custom format (also added “lbs”):



* Custom Formats won’t change underlying cell data.
* Custom Formats will propagate to graphs and reports you create.
* Introduce Preview area, helpful during formula creation.
* Tip: Numeric data formats right-justified, character data formats left-justified.
* Oft-used functions can be added as Favorites in the searchable functions list.

Goals for ‘BMI\_current’, ‘BMI\_prior’, ‘%Change BMI’ formulas:

* Use intermediate columns to create more complex formula
* Data cleanup to not use ‘outlier’ data in the formula
* Debugging formulas using Formula Editor and the JMP log window.
  + ‘Show()’ values in the log file for rows with ‘missing’ data
* Introduce JMP Scripting Index to look up function syntax
* Create custom functions using a template provided by JMP. Your new function shows up in the searchable functions list and the Scripting Index.
* Introduce and use local variables and a table variable
  + Local used only in that formula
  + Table variables can be used in any column formula in that table



Here’s the JSL for this larger formula (with debug log files messages):

t1 = (:BMI Current - :BMI Prior) / :BMI Prior;

If( Is Missing( t1 ),

Show( :name );

Show( :BMI Current );

Show( :BMI Prior );

Show( "-------" );

);

t1;

Goals for ‘*SSN*’ and ‘*Weight String’* string manipulation formulas:

* Example of data cleanup using ‘*Substitute()*’:



* Capture ‘last 4’ of SSN using ‘*Substr()*’:



* Introduce an example using ‘*Munger()*’:



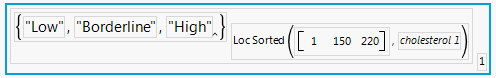
* Convert string to number



* Hint of Strings (left-justified) vs. Numbers (right-justified) in data tables

Goals for ‘*Cholesterol 1-3’*:

* Show how to do Column Standardize
* Introduce ‘*Loc sorted()’* as way to quickly categorize and do lookup tables



Here’s JSL for this more complex formula:

{"Low", "Borderline", "High"}[Loc Sorted( [1 150 220], :cholesterol 1 )][1]

There is a great example in more detail using *Loc Sorted()* in a post by Brady Brady on the JMP Community.   
<https://community.jmp.com/t5/JMP-Blog/Lookup-tables-in-JMP/ba-p/30434>

Goals for ‘Date’ manipulation:

* Introduce ‘Date’ data type. Number represents number of seconds since 1904.
* Show dates with different built-in JMP Date/Time formats
* Discuss ‘Suppress Eval’
* Generate MM/YYYY date from *:Random Dates*, keeping date data type.
  + Date MDY( Month( :Random Dates ), 1, Year( :Random Dates ) )
* Generate YYYY-WW (week of the year) as a STRING



* Calculate difference between 2 dates (in weeks) using ‘*Lag()’* function



* Generate new date column that is +5 hours



* Check out Brady Brady’s Data Table Tools Add-In which includes MANY tools for parsing and creating date data as well as many other JMP tools.

<https://community.jmp.com/t5/JMP-Add-Ins/Data-Table-Tools-Add-in/ta-p/28582>

**Conclusion**

We hope you’ve learned the basics of how to build formulas and debug them. Remember to use all of the help that JMP provides with the Scripting Index (under the Help menu). We also highly recommend joining the JMP User Community at <https://community.jmp.com>. Chances are if you have a question about how to tackle a specific task in JMP, someone else has already had that issue and has posted a solution for the community to use.

**Appendix**

* Invoking the Formula Editor
* Formula Editor Concepts
  + Calculator Buttons
  + Undo Redo
  + Clear Formula
  + Searchable Functions List
  + Columns List
  + Displaying Matrices
  + Tiny Editor and Full Script Editor
  + Custom Formats
* Explaining User-Defined Custom Functions
* Another example from the JMP Community

**Invoking the Formula Editor**

There are several ways to bring up the editor, depending on whether it’s an existing formula or not. For existing formulas, you can click on the “+” sign next to the column name in the Data Table. The “+” indicates that there is a column formula which is the source for the data in this column. This brings up the Formula editor with one click.

The following options to invoke the FE will bring up the editor with a blank workspace if there is not currently a formula in that column. Otherwise it will edit the current formula, just like hitting the “+”.

1. Bring up the Column Info… dialog (either right mouse on column title in the data table -> Column Info... OR you can double-click on the column title in the data table), then select Formula, then click Edit Formula button. Working through the Column Info… allows the user to set Column Name and other properties as well as edit the formula.
2. Second way to bring up FE is to context click on the column title in the Data Table-> select Formula… to bring up the editor directly.

**Introductions to Formula Editor concepts**

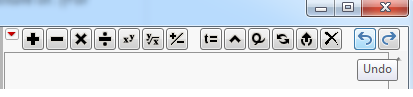
The rest of this section introduces some features and concepts of the Formula Editor. These features will be used as we explore how to invoke the editor and then how to create formulas.

**Calculator Buttons**

Simple calculations are easily accessible in the form of calculator buttons available at the top of the editor workspace.

**Undo Redo**

Undo and Redo are a longtime request of JMP users, and JMP 13 finally implemented a full Undo/Redo stack. Undo and Redo buttons are available along the top of the Formula Editor with the other buttons. You can also use Control-Z, Control-Y (or Command-Z, Command-Shift-Z on Mac) if you like to use the keyboard. See blue Undo and Redo buttons below.

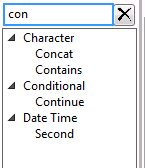


**Clear Formula**

The ‘Clear’ button has been relocated to the top button row. You can still clear the workspace, resetting it back to ‘No Formula’ using the convenient new ‘X’ button. Don’t worry, if you pressed this accidentally, you can Undo and retrieve your work.

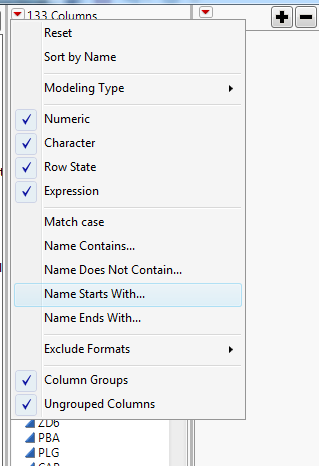
**Searchable Functions List**

The functions tree/list is expandable, as mentioned above. The new live filtering makes it very easy to find functions you know you want to use, but may not know which category they are in. For example, typing **con** would display this filtered function tree:

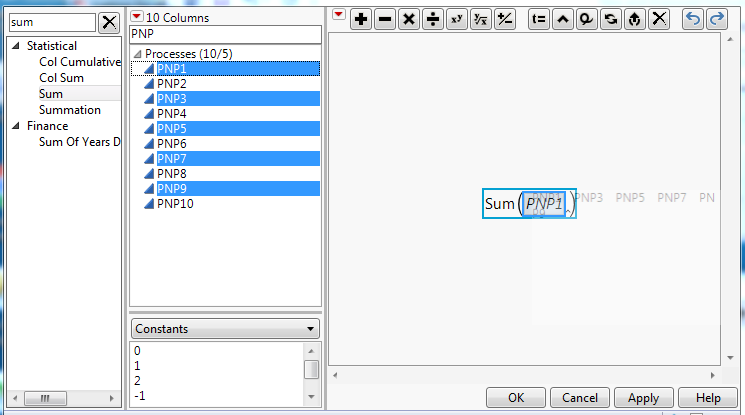


**Columns List**

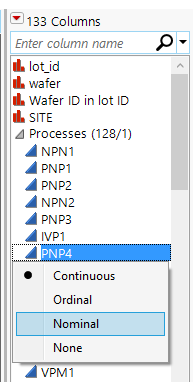
The columns list is a “real” columns list, as available in most launch dialogs, with the ability to filter and search columns. For example, Name Starts With… ‘PNP’.



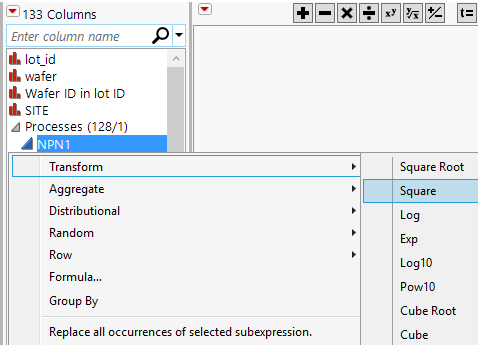
You can now select multiple columns (Shift-click or Ctrl-click), then drag and drop to populate a formula.



You can also change the modeling type of a column quickly and easily,

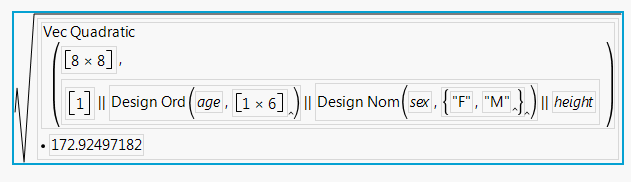


or create a column transform.

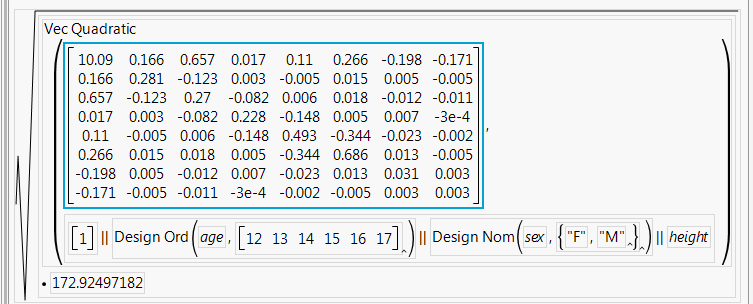


**Displaying Matrices**

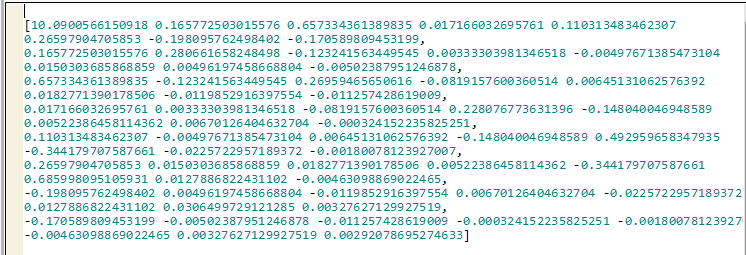
The JMP Formula Editor can now display matrices nicely inside of the editor. You can determine the size of the dimensions that are shown from the red triangle menu. Matrices are abstracted to show only the size if dimensions exceed the display size.



For this example, if ‘Max Matrix Size to Show…’ is increased to 8, then you would see actual matrix values. These values may be truncated,

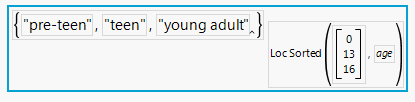


but you can always double-click to see the full matrix in the built-in JSL editor.

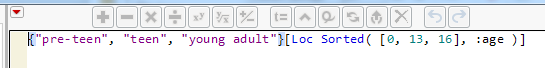


**Tiny Editor and Full Script Editor**

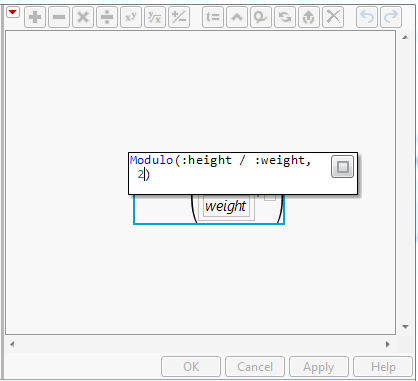
When you want to switch to writing the JSL yourself instead of using the point-and-click interface, this is easy to do using the built-in JSL editor, complete with the coloring of functions and tooltips when you want and expect them.



Double-clicking on this formula displays the JSL in a full-screen script editor.

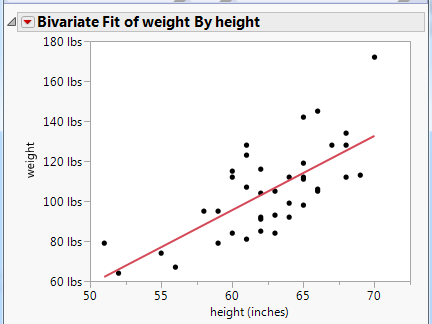


Editing smaller formulas will use the tiny editor.

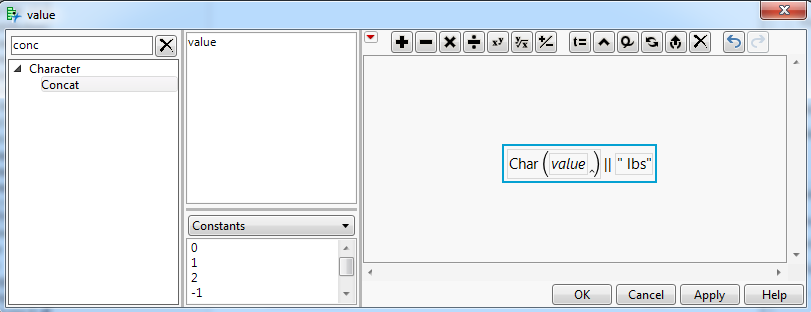


**Custom Formats**

Custom formats are a new feature in JMP, which make use of the new Formula Editor in order to define them. For example, sometimes you would like your axis labels to display units instead of just the values. Below, see the weights are displayed with ‘lbs’ units.



In order to define this view, right-click on the axis and a unique instance of the Formula Editor will appear. Define the unit you want by using the concat() function to append “ lbs” to the end of the value. Here is the corresponding formula to add ‘lbs’ to the weight values.



Custom formats may be added to Axis Properties or to column formats directly. Please see the accompanying “Step By Step Instructions for Demo.docx” for detailed instructions.

NOTE: Care should be taken if the actual ‘value’ is changed in the custom format. It will display this new value, but won’t change the underlying cell’s value. This can be tricky.

**User-Defined Functions**

JMP has the capability to define user-defined Custom functions. These functions can then be used in the Formula Editor’s function list, in JSL scripting, as Custom Formats and as Custom Transforms. Note that these Custom functions can only be used in the context of the current JMP session. Restarting JMP will wipe out previously defined Custom Functions. They must be defined and added each session. An easy way to do this is to define your functions in an Add-In that runs each time you start JMP.

To define a custom function, use the ‘New Custom Function()’ function. Once a function is defined, you can add/activate it to the current running instance of JMP using the ‘Add Custom Functions()’ function. Similarly, you can remove/deactivate a function using the ‘Remove Custom Functions()’ function.

New Custom Function() requires 3 parameters: namespace, functionName, functionDefinition. Additional messages may be passed to define:

1. Formula Editor values: Formula Category, Result type, Parameter(s)
2. Custom Transform values: Transform Category
3. Custom Format values: Custom Format Category
4. Scripting Index values: Description, Prototype, Scripting Index Category, Example(s)

**Example 1: Custom Function**

As the 1st example, let’s create a completely specified function (including Scripting Index information and hint text). We will define the function first, then add/activate it.

funcDef=function({x, y=10}, x+y);

description="This function adds 2 values. If only 1 argument is specified, the 2nd argument defaults to 10";

// Examples can be multiple statements inside expr()

ex2=expr(

dt = Open( "$SAMPLE\_DATA/Big Class.jmp" );

myNameSpace:AddTen(:age[4], 20)

);

parmHint1="x value";

parmHint2="<y=10>";

//create function using variables

newAdd=New Custom Function("myNamespace", "AddTen", funcDef);

newAdd << Description(description);

newAdd << Prototype("myNameSpace:AddTen(x,<y=10>)");

newAdd << Example("myNameSpace:AddTen(4)");

// expr() Examples need to be passed inside NameExpr()

newAdd << Example(NameExpr(ex2));

newAdd << Parameter("Number", parmHint1);

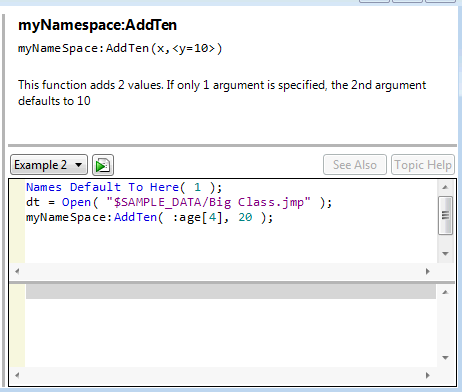
newAdd << Parameter("Number", parmHint2);

newAdd << Formula Category( "NumberStuff" );

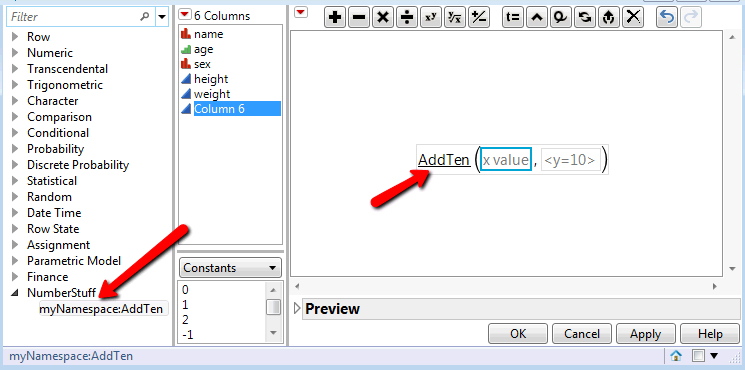
//add myNamespace:AddTen function to system

Add Custom Functions(newAdd);

Executing the above JSL will write “Deploying function: myNamespace:AddTen” in the JMP log. Searching for this function in the Scripting Index would show the results of using the Description, Prototype, and Example messages (see screenshot below):



The Formula Category message results in a new category in the formula editor’s list of formulas. Custom Functions are shown with an underline in the Formula Editor. Also see the hint text from the Parameter messages (see below):



**Example 2: Custom Transform**

As another example, let’s create a simple Custom Transform to multiply a value by 1000. We will embed the function definition inside of the Add Custom Functions() call.

Add Custom Functions(

New Custom Function(

"custom", // Namespace in which new function resides

"x1000", // Function name. Completely scoped name is "custom:x1000"

// Function definition.

Function( {x}, x\*1000 ),

// Optional message to enables this as a Custom Transform

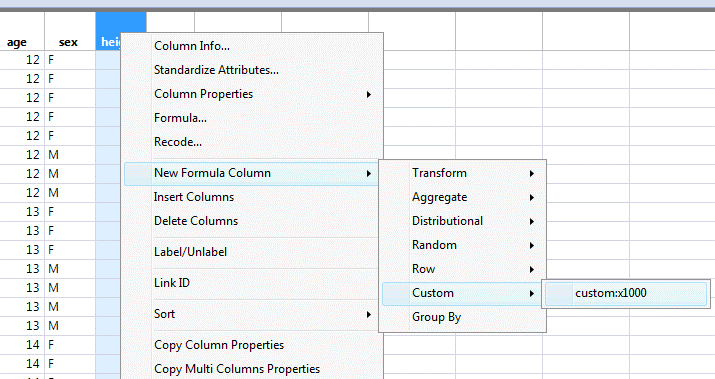
<< Transform Category( 1 )

)

);

Executing the above JSL will write “Deploying function: custom:x1000” in the JMP log. Searching for this function in the Scripting Index would only show that ‘custom:x1000’ is an existing function. No other description of any kind exists for this new function.

Now, creating a new column using the newly added transform would look like:



**Example 3: Custom Format**

The last type of Custom Function is a Custom Format. This is created using the same ‘New Custom Function()’ function, but also specifying the ‘Custom Format Category(1)’ message. Here’s an example that multiplies an input variable by 2, but also displays ‘(x2)’ after the value. Note that in this example we have enabled this function to be persisted as a Custom Format as well as in the formula category “My New Category”. This would allow you to use this in a column formula or as a Custom Format.

// this section adds a New Custom Function, specifying it as a Custom Format.

// This will show "(x2)" after the value in the DT

Add Custom Functions(

New Custom Function(

"myNamespace",

"Times2",

Function( {inputVar},

{Default Local},

char(inputVar\*2) || " (x2)"),

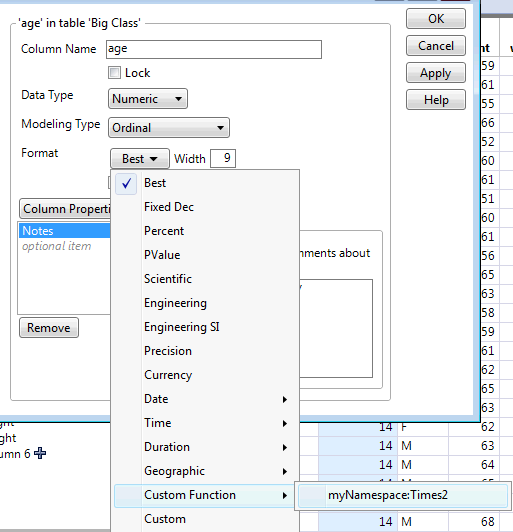
<<Description( "Multiply input by 2. Display (x2) after new value" ),

<<Formula Category( "My New Category" ),

<<Custom Format Category( 1 ) )

);

By specifying the ‘Custom Format Category(1)’ message, this format now persists under the ‘Custom Function’ category in the format menu. Here’s what this looks like when applying this new ‘Times2’ custom format in the Column Info dialog.



**Tip: Make your custom functions persist across JMP sessions**

Option 1) set up jmpStart.jsl. If you want to run the same script every time you start JMP, name it jmpStart.jsl and place it in one of the following folders, as appropriate for your operating system. When JMP starts, JMP looks for the jmpStart.jsl script in these folders in the order in which they are listed here. The first one that is found is run, and the search immediately stops.

Note: Some path names in this section refer to the “JMP” folder. On Windows, in JMP Pro, the “JMP” folder is named “JMPPro”. In JMP Shrinkwrap, the “JMP” folder is named “JMPSW”.

Windows:

1. C:\Users\<username>\AppData\Roaming\SAS\JMP\<version number>
2. C:\Users\<username>\AppData\Roaming\SAS\JMP

Macintosh

1. /Users/<username>/Library/Application Support/JMP/<version number>
2. /Users/<username>/Library/Application Support/JMP

Option 2) create a JMP add-in. A JMP add-in is a JSL script that you can run anytime from the JMP Add-ins menu. You can create submenus to group your JMP add-ins and have many levels of menus, if desired. Experienced JSL script writers can create scripts that extend JMP in many ways. For best instructions, see the JMP Help “Compile Scripts into an Add-in Using Add-in Builder.”

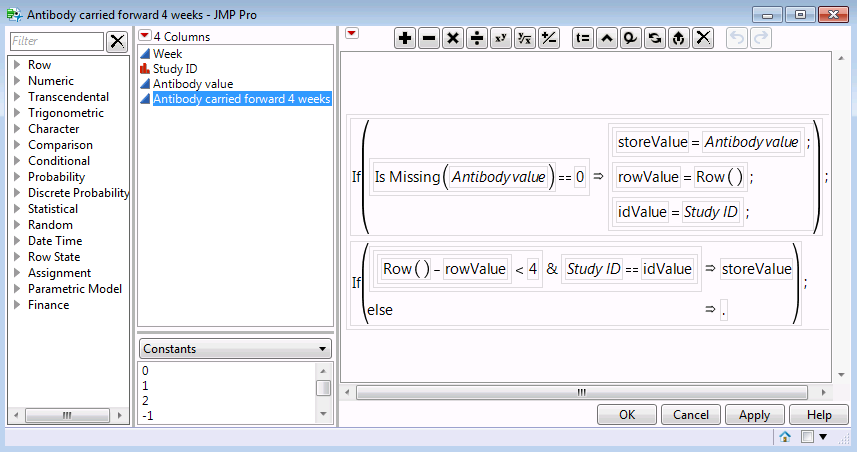
**Another Example from the JMP Community**

This problem and solution are based upon this example in the JMP User community: <https://community.jmp.com/t5/Discussions/Iterating-within-a-Column-Formula/m-p/33681#U33681>

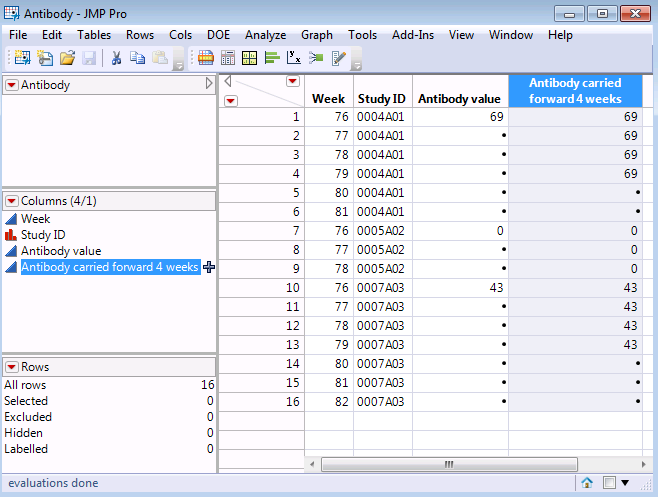
Re-stated problem statement:

*User wanted to copy the value from another Column (Antibody value) and paste it into the next X rows. If ‘Antibody value’ is not missing, copy it down 4 rows into column ‘Antibody carried forward 4 weeks’. Only copy the value if the Study ID has not changed.*

This solution is based upon a response from JMP Community Super User, ‘txnelson’. As he mentioned in the post, “… JMP is going to process down through all rows, it is just simpler to let JMP do the looping.” See the data table and formula screenshots below.

****

**Screenshot: More complex formula working across rows**

****

**Screenshot: Antibody data table with formula results**