Paper 169-30

# JMP® vs SAS® Overview Along with a JSL® Guide for the SAS Programmer

Dr. Andy Mauromoustakos and Kevin Thompson
Agricultural Statistics Laboratory, 101 AGRX, U of Arkansas, Fayetteville, AR 72701

## ABSTRACT

First we will provide an overview that provides SAS users a quick comparison among SAS and JMP. Then we will focus our attention on providing SAS programmers with a quick guide for JSL (JMP's Scripting Language). JSL was added in JMP version 4 and is, in our opinion, the boldest new feature delivered for some time. It, for the first time, provides users a powerful language that gives knowledgeable JMP users the opportunity to more easily perform routine tasks and extend JMP beyond its already broad capabilities. We will illustrate things by sketching the construction of an application for the generation of Honeycomb field designs used for crop breeding by geneticists performing plant selection. We will compare JMP and SAS versions of the application from both the programmer's and the user's perspective. What we are hoping to show SAS users that JSL can be one of the new learning challenges in their wish list.

## INTRODUCTION: A QUICK COMPARISON OF JMP TO SAS

### SYSTEM OVERVIEW

| *JMP* | *SAS* |
|---|---|
| Limited to data manipulation, exploration and statistical analysis. "Similar" to what SAS was trying to be in the 1970's. | A full data manipulation and presentation package. Statistics in currently only one component |
| Unlike SAS then it is designed to have a point and click, menu driven interface with limited programmability. Programming is tightly bound to the JMP windowing environment. | Program statement and command interface oriented. Various point and click interfaces have been added on (Assist, SAS Desktop, Analyst, and Enterprise Guide). Programs are often independent of the windowing environment. |
| Interactive, graphically oriented results. | Static document oriented results |
| Two programming languages : JSL and Visual Basic | Multiple interconnected programming Languages: SAS Language, Data Step Language, IML, SCL, Macro Language |
| Under rapid development. Components and programs may need major maintenance between version changes. | Very Stable. A SAS program written under MVS in the 1980's will probably need little modification to run under the current . |

### SYSTEM COMPONENTS

The components in JMP (Data Tables, Platforms and Reports) are considered to be "objects". They are manipulated by sending "messages" to them. The object/message structure only becomes apparent when writing JSL scripts.

| *JMP* | *SAS* |
|---|---|
| Data tables | SAS Data Sets |
| Value Labels (Ver 5), inside Data Tables | Formats, in separate catalog |
| Platforms | Procedures |
| Reports<br>Reports are highly Interactive Information maybe added and removed after a report is generated. Most items may be converted to Data Table at will. | Results<br>SAS output is static. It is usually necessary to re-run procedure to modify the contents of the out put or convert the output to a Data Set |
| Layout/Journals<br>Allows customized arrangement of Report items. | The Output Delivery System.<br>The version 9 Document destination allows some customization of arrangement of Results. |
| JSL scripts | SAS programs |

## JSL (THE JMP SCRIPTING LANGUAGE) WITH REFERENCES TO SAS

JMP is optimized for interactive exploration and analysis. Scripting is a way to tell JMP how to do something that would normally be accomplished by way of mouse actions. When you understand the "best" way to interactively accomplish a task, then you are close to the "best" script. Imagine that you must speak or write the instructions for the same steps that you would normally communicate to JMP with the mouse. Most JSL scripts are small and simple, similar to single Assignment statements or PROC steps in SAS.JSL provides basic program flow control and dialog box creation elements to allow for more complex applications. In summary JSL is the JMP native programming language. It is similar to a combination of the SAS Data Step language, AF, Macro and IML languages.

### WHAT YOU CAN DO WITH JSL

Almost any task or analysis that you can perform in JMP can be coded in JSL. In fact, many of the tasks commonly performed in JMP, such as selecting an item in the JMP menu to invoke a Platform to perform an analysis, are internally converted to an equivalent JSL Script. Most JSL scripts are small stored programs that allow you to repeat an analysis without having to repeat the entire sequence of mouse clicks necessary to perform the analysis.

### BASIC USES OF JSL

The two most common uses of JSL are in Column Formulae and as a stand-alone script.
A Column Formula is a JSL script stored within a Data Table column. It is generally used to set the value of a Data table column based upon the existing values of other columns. This is similar in function to a SAS Assignment statement.

A stand-alone script more closely matches a SAS program. Stand-alone scripts may be stored in a separate file on the computer or in a Table Property embedded within a JMP Data table. Common uses of stand-alone scripts are:

- Create and modify Data Tables (Similar to Data Step or IML program)
- Create and modify Data column values—Column Formulae (Similar to Data Step or IML assignment statement.
- Invoke and modify Platforms (Similar to PROC step programs)
- Create custom Reports (Similar to DATA_NULL_; step)
- Create a new application (Similar to AF, IML, and Data and Proc Steps)
- Combine all of the above (Similar to Macro programming)

### OBJECTS AND MESSAGES

Most components in JMP are considered to be objects. JSL often involves sending messages to these objects. Messages tell the objects what to do and how to do it.

Reports display the results of Platforms, which analyze data in Data tables. These objects are dynamically linked, so that modifying one object often imposes changes in other objects.

- Messages to Data tables allow for the dynamic creation and deletion of new rows, columns and table variables.
- Messages sent to platforms allow you to change the nature of the analysis.
- Messages to reports allow you to dynamically change the layout of the report.

JSL utilizes the object orientation approach that makes applications that are easier to understand, sometimes reduces time spent in application development, makes applications more easily maintainable and allows for better scalability.

### STRUCTURE OF JSL

JSL is simple. The basic unit is an expression, usually in the form of a function. All expressions, including the Assignment expression and the message sending expression, are equivalent to and can be expressed as a function. While a JSL script may appear to be a sequence of statements terminated by a semi-colon, the semi-colon is actually an expression separator and not a statement terminator. The semicolon is equivalent to the GLUE() function. Using it as an expression terminator can sometimes causes a syntax error. A complete JSL script can be considered as a sequence of nested functions.

### A FIRST LOOK AT JSL

The first script a JMP user is likely to see is a Column Formula, used to assign data values to one column based upon the values in other columns.

In addition, every JMP Platform provides a method of saving a JSL script that will repeat the analysis. This JSL script may be stored in a separate file, or in a Table Property inside the Data table itself. Beginning with Version 5.1, this script may also include some user modifications to the platform's report layout. Most JSL scripts begin with such generated scripts. The programmer can then cobble together modified collections of theses scripts to create a more complex application.

### EXPRESSIONS

The basic structure of a JSL script is the expression. Most expressions have the form of a function reference (e.g. `Function( arg1, arg2, … )` ). Even such common expressions as a variable assignment , `y=1+2` , is equivalent to nested Assign() and Add() functions:, `ASSIGN(y,ADD(1,2)).`

### THE SEMI-COLON (;)

Unlike SAS, the semi-colon is an expression separator, equivalent to the Glue() function, and not a statement terminator. Therefore, unexpected semi-colons can often cause unexpected syntax errors. A quick solution is to use a dummy expression as a placeholder:

```
        IF(  x> 1,
             y=”Yes”  ;   // This looks like an assignment statement
             0 )  ;       // The dummy expression avoids problems with the ;
        0                 // Another dummy expression
```

### VARIABLES

As in other languages, variables are used to store data values. JMP defines three types of variables.

- Script, or JSL, variables store data values used by a JSL script. Extra care must be used when a script variable is used in a Column Formulae.
- Column variables sometimes refer to an entire column in a data table and sometimes to a particular value on one row in the column. These variables are commonly used in scripts stored in column Formulae and in stand-alone scripts. Column variable are commonly used in both Column Formulae and in stand-alone scripts.
- Table variables are similar to constants stored with a table. These variables are immediately available to JSL scripts stored in Column Formulae, but not to stand-alone scripts.

Unfortunately, all three types of variables follow the same naming convention. The actual evaluation of a variable depends on the context where it is used, and may depend upon scripts and Data tables used earlier within the JMP session.

JMP does provide the colon ( : )and double-colon ( :: ) scoping operators to aid in the discrimination between Script and Column variables. While JSL does allow for a locally scoped script variable, a variable whose value is set for a limited duration and reverts to its previous value at the termination of a section of code, actually doing this for more that a few variables is cumbersome. It is best to assume that a script variable is active for the entire JMP session. A personalized naming convention within JSL scripts is recommended to avoid variable name collisions.

Variable names may be composed of any number of characters (a-Z, a-z), digits (0-9), white spaces (space, tab, new line and new page characters), and selected symbols (underscore, apostrophe, percent sign, period, and back slash). The variable name must start with either a character or the underscore. Capitalization and white space is preserved but ignored. For example, the variables `Foo Bar` and `foobar` are considered to be equivalent.

Column variables may only contain one of three types of data, numeric, character strings and row states.
Scripts variables may contain many different types of data: scalars, character strings, pointers to the various JMP objects (Data Tables, Columns, Platforms and Reports), matrices, lists, and entire expressions.

### MODULARITY AND THE EXPR() FUNCTION

JSL provides no modularity. Because the arguments of many functions may become very long, it is best to store a lengthy section of JSL code in a variable, to be evaluated later in the script.

```
   ::exprThen = EXPR(
                     . . .;  // A long sequence of expressions
                   0);       // ended by a dummy expression
   ::exprElse = EXPR(
                     . . .;
                   0);
   IF( condition, ::exprTHEN, ::exprElse );
```

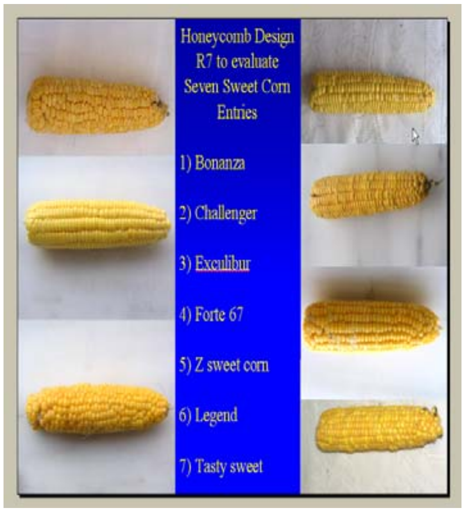
This same idea may even be applied to a large JSL application

```
   // The Honeycomb JSL application
   // Define MAIN expression
   ::MAIN= Expr
       ::HDGSetDefaults;
       ::HDGGenerateDesignTable(::numHDGMinR, ::numHDGMaxR );
       ::HDGDisplayDesignPlatform;
     0); //End::MAIN

   // Module definitions
   ::HDGSetDefaults=Expr(  . . . ) ;
   ::HDGGenerateDesignTable= Function( {MinR, MaxR} , . . . ) ;
   ::HDGDisplayDesignPlatform-Expr( . . . ) ;

   // Run Program
   ::MAIN;
   0       // Return 0
```

## THE HONEYCOMB APPLICATION

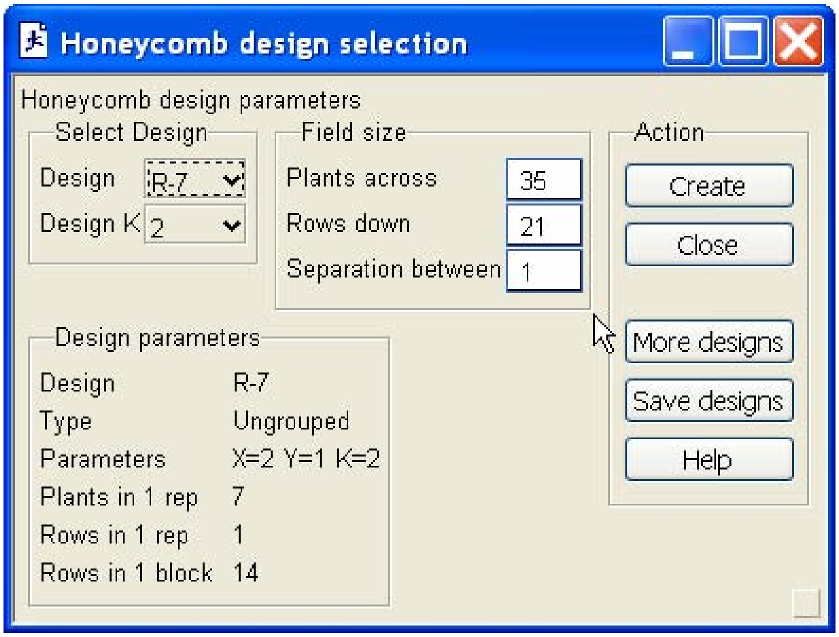
To illustrate similarities and differences between JSL and SAS, and to make some overall comparisons between the two, we will explore the Honeycomb application. Honeycomb designs (Fasoulas and Fasoula 1995) are a set systematic, hexagonal pattern designs with at a set plant-to-plant spacing. They were developed to speed up the process of selecting individual plants among genetic entries. These designs accomplish their goal by incorporating spatial field variability and environmental diversity. The Honeycomb application produces a set of potential Honeycomb Designs and provides the user with the ability to pick a suitable design depending on his or her needs and circumstances. The user may also explore the field layout to visualize the data. It is worth pointing out that the generation of Honeycomb Designs used in the plant breeding and selection and does not exist as a standard design in either the SAS or JMP Design of Experiments platforms.

A Honeycomb design was proposed to the best individual plants from among 7 varieties of sweet corn. These individual plants will be used in further plant breeding in order to improve the lines. For our example, an R-7 (X=2, Y=1, K=2) honeycomb design was selected with 7 corn entries arranged in 21 rows of 35 plants. The plants are arranged in a hexagonal arrangement where every plant is surrounded by plant of a different variety.



## THE HONEYCOMB JMP APPLICATION

The entire Honeycomb JSL application is contained in the file Honeycomb.jsl. To run the application, it is only necessary to open and run the script. Upon execution, the application displays the “Honeycomb design selection” window where the user may select the desired design and field size.



Honeycomb design selection

Honeycomb design parameters

Select Design

Design: R-7

Design K: 2

Field size

Plants across: 35

Rows down: 21

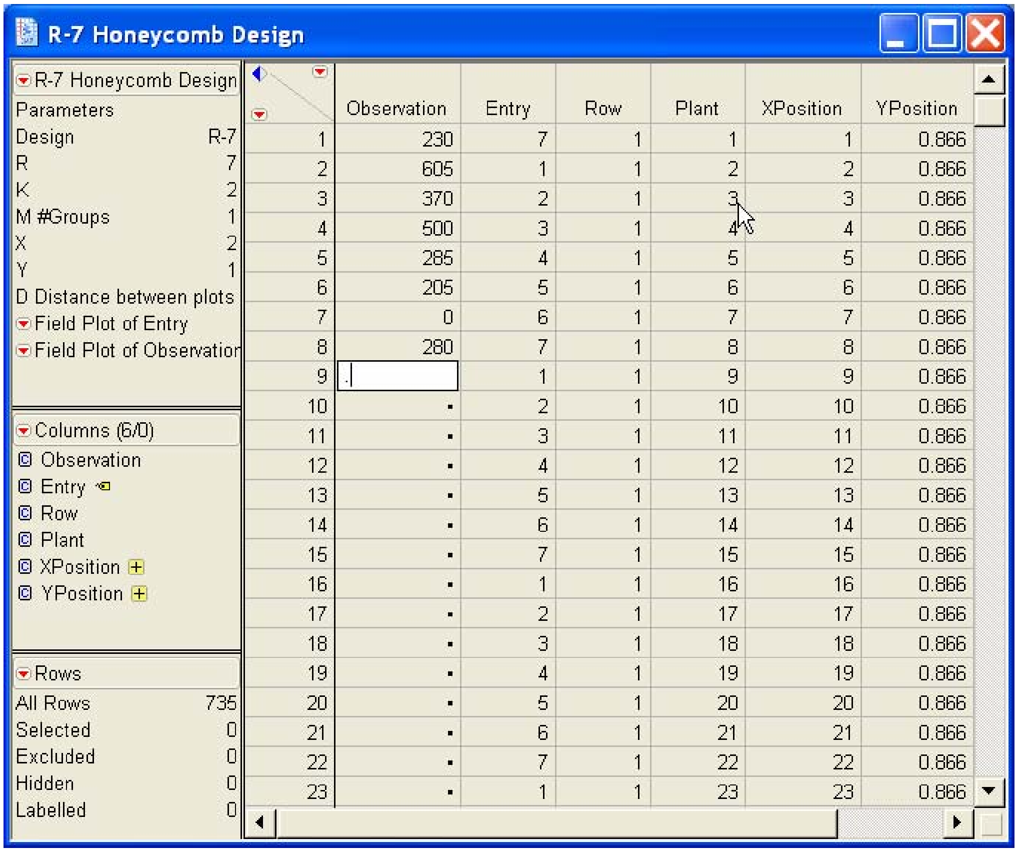
Separation between: 1

Design parameters

| | |
|---|---|
| Design | R-7 |
| Type | Ungrouped |
| Parameters | X=2 Y=1 K=2 |
| Plants in 1 rep | 7 |
| Rows in 1 rep | 1 |
| Rows in 1 block | 14 |

Once the use has selected the desired design and field size, it is only necessary to press the “Create” button to generate a JMP table containing the planting plan. Additional buttons allow for the saving of all of the designs with their creation parameters as a JMP table, and the generation of additional Honeycomb designs suitable of experiments with differing number of entries.

Upon completion of the experiment, the user may use standard JMP techniques to enter the yield, or other, data.


R-7 Honeycomb Design

R-7 Honeycomb Design
Parameters

| | |
|---|---|
| Design | R-7 |
| R | 7 |
| K | 2 |
| M #Groups | 1 |
| X | 2 |
| Y | 1 |
| D Distance between plots | |

Field Plot of Entry
Field Plot of Observation

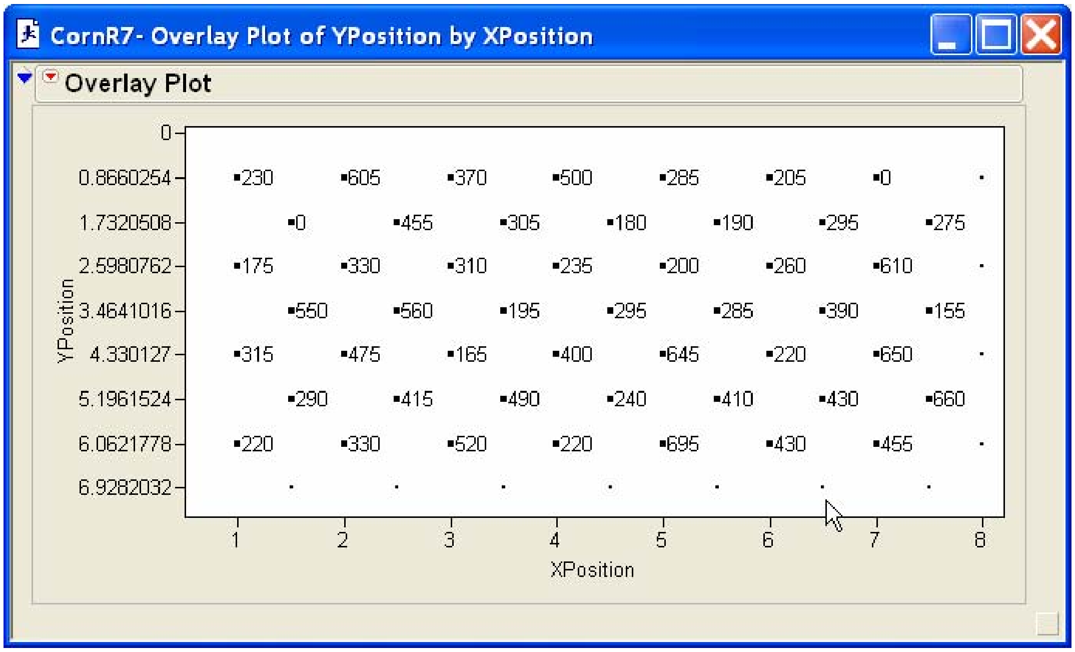
Columns (6/0)
Observation
Entry
Row
Plant
XPosition
YPosition

Rows

| | |
|---|---|
| All Rows | 735 |
| Selected | 0 |
| Excluded | 0 |
| Hidden | 0 |
| Labelled | 0 |

| | Observation | Entry | Row | Plant | XPosition | YPosition |
|---|---|---|---|---|---|---|
| 1 | 230 | 7 | 1 | 1 | 1 | 0.866 |
| 2 | 605 | 1 | 1 | 2 | 2 | 0.866 |
| 3 | 370 | 2 | 1 | 3 | 3 | 0.866 |
| 4 | 500 | 3 | 1 | 4 | 4 | 0.866 |
| 5 | 285 | 4 | 1 | 5 | 5 | 0.866 |
| 6 | 205 | 5 | 1 | 6 | 6 | 0.866 |
| 7 | 0 | 6 | 1 | 7 | 7 | 0.866 |
| 8 | 280 | 7 | 1 | 8 | 8 | 0.866 |
| 9 | . | 1 | 1 | 9 | 9 | 0.866 |
| 10 | • | 2 | 1 | 10 | 10 | 0.866 |
| 11 | • | 3 | 1 | 11 | 11 | 0.866 |
| 12 | • | 4 | 1 | 12 | 12 | 0.866 |
| 13 | • | 5 | 1 | 13 | 13 | 0.866 |
| 14 | • | 6 | 1 | 14 | 14 | 0.866 |
| 15 | • | 7 | 1 | 15 | 15 | 0.866 |
| 16 | • | 1 | 1 | 16 | 16 | 0.866 |
| 17 | • | 2 | 1 | 17 | 17 | 0.866 |
| 18 | • | 3 | 1 | 18 | 18 | 0.866 |
| 19 | • | 4 | 1 | 19 | 19 | 0.866 |
| 20 | • | 5 | 1 | 20 | 20 | 0.866 |
| 21 | • | 6 | 1 | 21 | 21 | 0.866 |
| 22 | • | 7 | 1 | 22 | 22 | 0.866 |
| 23 | • | 1 | 1 | 23 | 23 | 0.866 |

Embedded in the JMP table is a table Property, “Field Plot of Observations” containing a script that, when executed, displays a field map of the observational data. An edited version is shown below.


CornR7- Overlay Plot of YPosition by XPosition

Overlay Plot

## THE HONEYCOMB SAS APPLICATION

The Honeycomb SAS application is contained in the file Honeycomb.SASMacro. Unlike the JSL version, the SAS version only contains a set of SAS macros that the user must invoke in a separate program. While it is possible to create a “wrapper” environment using SAS\AF to make the execution of the macros easier, such a program would be stored in a binary SAS catalog which may not be easily ported to other versions of SAS running under other operating Systems.

To select a design, run the following SAS program to display a list of possible Honeycomb designs, along with the design creation parameters.

```
%Include “Honeycomb.SASMacro”  / NoSource2;
%HoneycombGenerateParameters(Entries=3, OUT=DesignList);
Proc Print Data=DesignList; Quit;
```

The following program creates a SAS data set containing a plot plan for the R-Design.

```
%Include “Honeycomb.SASMacro”  / NoSource2;
%HoneycombGenerateDesign(X=2,Y=1,K=2, NRows=21,NPlants=25,  Out=R7Honeycomb);
Proc Print Data=R7Honeycomb; Quit ;
```

To enter the yield data “you are on your own”. Different individuals have different preferences on how to edit a SAS data set. Some prefer to export to a SAS data set to a spreadsheet, make additions there, and then re-import it back to SAS. Others may prefer to use the SAS ViewTable window or even use PROC Insight.

You may plot a field map of the data with PROC GPLOT

```
Proc GPlot Data=R7Honeycomb;
     PLOT YPosition * XPosition= [ …  ];
Quit;
```

## SAS VS JMP HONEYCOMB COMPARISON FROM THE USER PRESPECTIVE

JMP provides a nicer environment in which to run the Honeycomb application.

| ***Application Tasks*** | ***JMP*** | ***SAS*** |
|---|---|---|
| Generate a Plant Plan | Use the “Create” button The Designs parameters are obtained from the User Interface. | Write a SAS program<br>`%HoneycombGenerateDesign(X=#,Y=#,K=#, NRows=#,NPlants=#,  Out=sasds);`<br>`Proc Print Data=sasds);`<br>`Quit;`<br>The design parameters must be manually entered. |
| Enter Observational Data | Use standard JMP tools. E.G. Edit Data Table, Table Join. | Use standard SAS tools E.g. Programming, DATA/MERGE, View Table, FSEDIT |
| View a field map | Run the embedded “Field Plot” script.. | Write your own SAS program. |

**SAS VS JMP HONEYCOMB COMPARISSON FROM THE PROGRAMMER PRESPECTIVE**
In our experience it is easier to write and debug applications with SAS. Most of the time, the SAS code is also shorter. JMP does haves some capabilities, such as application created dialog boxes, that are not easily available in SAS.

| ***Application Creation*** | ***JMP*** | ***SAS*** |
|---|---|---|
| Create User Interface | Can be created by a program<br>User Interface is stored in JSL code. | Old Method: AF<br>New Method: SAS AppDev Studio<br>AF: created at design time using a GUI environment<br>User Interface stored in SAS catalog may have troubles when is ported to other Versions of SAS or other Operating Systems. |
| Distributing the Application | Everything is stored in a single small ASCII JSL file. | SAS Macro code is stored in a single ASCII file Use standard SAS tools<br>User Interface if any is stored in a binary SAS Catalog that mist be defined to the SAS system before it can be used |
| Multiple Instances | Application must prevent a second instance from lunching.<br>Application control information is stored in global JSL variables which would be overwritten. | Macros run create SAS data sets and a listing and an output and then terminate. Difficulty to actually get two programs to run simultaneously is better |
| Debugger | Difficult. JMP provides no debugging tools.<br>Errors in JSL logs are difficult to find and interpret. | Better. SAS Logs provide more information.<br>Data Step debugger is unavailable |
| Variable name conflicts | Always a problem.<br>Data table variables and JSL are identical.<br>Unless care is taken a variable can refer to either a JMP table variable or a JSL variable. The default depends on any and all events that have occurred in the JMP session (previous scripts, previous open data tables, previous column formulas).<br>No time scoping of JSL variables the LOCAL() function exists but it is too difficult to use if you have more than few variables<br>ALL JSL variables are assumed GLOBAL. | Usually not a problem.<br>Dataset variables and macro variables look different which means that the programmer cannot confuse them.<br>Data variables are limited to Data Step and Proc Step code.<br>Macro variables defined within a macro are assumed LOCAL |
| Modularity | None<br>Any modularity has to be imposed on the JSL code by the programmer. | SAS codes naturally fits into distinct steps. |
| Stability of Environment | Under rapid development.<br>Components and programs may need major maintenance between version changes | Very Stable.<br>A SAS Program written under MVS in the 1980's will probably need little modification to run under the current version of SAS. |

## CONCLUSION

In our experience it takes similar lines of JSL code (most of the times the SAS code is shorter) to accomplish similar tasks in SAS by writing SAS macros although most experience SAS programmers you not trade the SAS programming environment to write in JSL. From the Programmer Perspective based on the Honeycomb application SAS wins on points. But from the user perspective on running the above application JMP wins hands down. We hope that eventually more SAS users will be willing to try and program applications with JSL.

## REFERENCES

Fasoulas, C. and V.C. Fasoula 1995, Honeycomb Selection Designs, *Plant Breeding Rev.* 13, 87-139.

*JMP® Scripting Guide*, Version 5 Copyright © 2002 by SAS Institute Inc., Cary, NC, USA.

Thompson K and A. Mauromoustakos. 2003. Making the Writing of JSL Applications Easier. 2003 Joint Statistical Meetings, Proceedings CD 7 pages.

## ACKNOWLEDGMENTS

Thanks to Dr. Dionysia Fasoula and her graduate student John Milonas for providing us with the corn data for their own field experiments.

## RECOMMENDED READING

Latest edition of the JMPer Cable Issue 16 Winter 2005 Special Scripting Edition. In particular you will find of interest the cover story article “Using SAS IOM Commands in JSL” by Matt Flynn, SAS Institute Inc.
The SAS IOM commands available in JMP Scripting Language (JSL) make it possible to connect directly from within JMP to a SAS server to retrieve data for further analysis within JMP. The script presented in this article focuses on library submission, highlighting the SAS IOM commands for connecting and executing SAS jobs, and returning the results back to JMP for further processing. We hope that eventually more SAS users will be willing to try and program applications with JSL (http://jmp.com/news/jmpercable/index.shtml).

## CONTACT INFORMATION

In case a reader wants to get a copy of the code please send correspondence to the author below.
Your comments and questions are valued and encouraged. Contact the author at:

Author Name: Kevin Thompson
Company: University of Arkansas
Address: 101 AGRX
City state ZIP: Fayetteville, AR 72701
Work Phone: (479) 575-6816
Fax: (479) 575-8643
Email: kthompsn@uark.edu