Paper 015-2012

Custom Analysis and Reporting with the JMP® Application Builder

Daniel Schikore, SAS Institute, Cary, NC, USA

ABSTRACT

The JMP Application Builder is a drag-and-drop environment for custom application development. Developers have access to the full capabilities of JMP, including more than 50 analysis and graphics platforms for in-memory analytics, as well as custom scripting with JSL to connect with SAS® or R for more powerful techniques. Custom, multi-window reports can be created interactively without having to write scripts at all. A flexible, hierarchical data filter allows the developer to apply filters to all or parts of a report. The developer can deploy a JMP Application for other JMP users to run, optionally encrypting the application with a password required to run the application.

INTRODUCTION

JMP is a desktop tool for in-memory data discovery and visual analytics. In addition to an extensive collection of a statistical and graphical capability, each release brings new ways to customize JMP. The JMP Scripting Language (JSL) includes support for creating and modifying strings and matrices, data tables, display box hierarchies (GUIs) and JMP platforms. The large collection of built-in statistical functions can be augmented through communication to external applications, including SAS and R.

A number of custom applications have been developed using JSL with SAS® Integration Technologies [Miclaus] [Watson]. JMP 9 added the ability to package and distribute scripts as an Add-In [Hill], making it easier to deploy custom applications within an organization or to a larger community. Add-Ins have the advantage that they embed new commands into the menus of JMP – a real convenience for scripts that are used repeatedly.

The JMP Application Builder is a new visual programming environment for JMP 10. You use intuitive drag-and-drop operations to create each application window. This eliminates a tedious part of the scripting process and makes it easy to experiment with various GUI configurations. You set properties on both the GUI objects and the Application objects through an integrated property table. Applications consisting primarily of JMP platforms often require no additional scripting, and can be parameterized to allow for runtime selection of columns to analyze. More complex applications can be created by adding custom JSL scripts to access the full capabilities of JMP, including integration with SAS and R. Applications can be saved to disk, stored as an attribute in a data table, deployed as an Add-In, or integrated into a higher level process script.

JMP APPLICATIONS

In the past, only experienced JSL programmers could customize many of JMPs powerful features. The purpose of the JMP Application and Application Builder is to provide the following capabilities:

- A lightweight framework to organize objects into building blocks for Applications
- Encapsulation to eliminate side effects
- Links to deployment mechanisms
- Integration with the JSL debugger

The JMP Application does not define or limit what an application can do. A JMP Application combines JSL scripting, display boxes, and JMP platforms into a designated workflow. The editing and run-time behaviors are controlled by three new JMP objects, the *JMP Application*, the *Module*, and the *Module Instance*.

The *JMP Application* consists of a JSL script, properties that are global to the application, and the definition of one or more Modules. JMP Application properties are shown in Table 1.

Name	The Application name is used as a default value for window titles and file names.
Table	In Edit mode, this data table is used to create the objects in the Application Builder. When the Application is run it will use the current data table, allowing you to run the same application against multiple sources.
Auto Launch	If any part of an Application has been parameterized (described later in Parameterizing Instant Applications), this Boolean property specifies whether the Application should automatically create a launch dialog box to prompt for and set the parameters. If you disable this, you are free to set

	the parameters through scripting.
Encrypt	Specifies that encryption should be used when saving an executable form of the application.
Run Password	An optional property can be specified to require the user of the application to enter a password when the application is run.

Table 1: Application Properties

The *Module* object is a design for a window in the application. It consists of a JSL script, a collection of display objects, and the module properties shown in Table 2.

Variable Name	This is the name of the Module in the Application namespace. It can be used to query or send messages to a Module from any script in the Application.
Title	This property is used as the window title when an instance of the Module is created at run time. Special strings `^TABLENAME` and `^APPNAME` can be included and will be substituted with the data table name or Application name.
Module Type	Specifies the type of window to be created. A *Dialog* or *Launcher* window has no menus or toolbars. A *Dialog with Menu* includes a menu bar. A *Report* window includes an auto-hide menu and toolbar.
Auto Launch	Setting this parameter will cause an Instance of this Module to be created automatically at run time. This is useful to enable certain classes of Applications to be created without scripting. For more complex application workflows, behavior can be controlled through the Application and Module scripts.

Table 2: Module Properties

The final object making up the Application Builder framework is the *Module Instance*. This is a run-time object representing the realization of a Module. While some Applications may create only a single instance of each Module, a custom Application can create multiple instances of the same Module based on user input, the data table, or other considerations.

RUNNING APPLICATIONS

It is helpful to understand how a JMP Application will behave at run time before trying to design an application. In a custom application, the scripts define the behavior of the Application. When an application is executed, the process flow is shown in Figure 1.

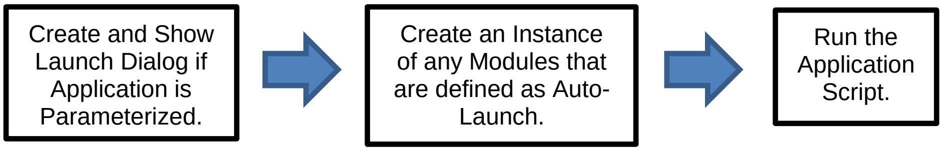

Create and Show Launch Dialog if Application is Parameterized.
Create an Instance of any Modules that are defined as Auto-Launch.
Run the Application Script.

Figure 1: Execution Flow for the JMP Application

The JMP Application is a very lightweight wrapper around the Module objects. The script gives the user full access to JSL functionality, but in general the work is done by the Modules that are created at run time. The Application script is used for definition of functions that are shared across multiple modules and set up processing before any analysis or display can be computed. For example, the script might confirm that the current data table is appropriate for the application or connect to SAS or R.

In a default configuration, one instance of each Module will be created automatically. This behavior can be changed through a combination of scripts and properties, which will be discussed in the section on custom applications. Each time a new Module Instance is created, the process flow is as shown in Figure 2.

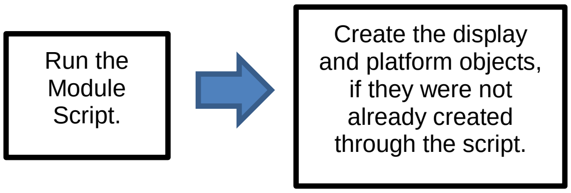
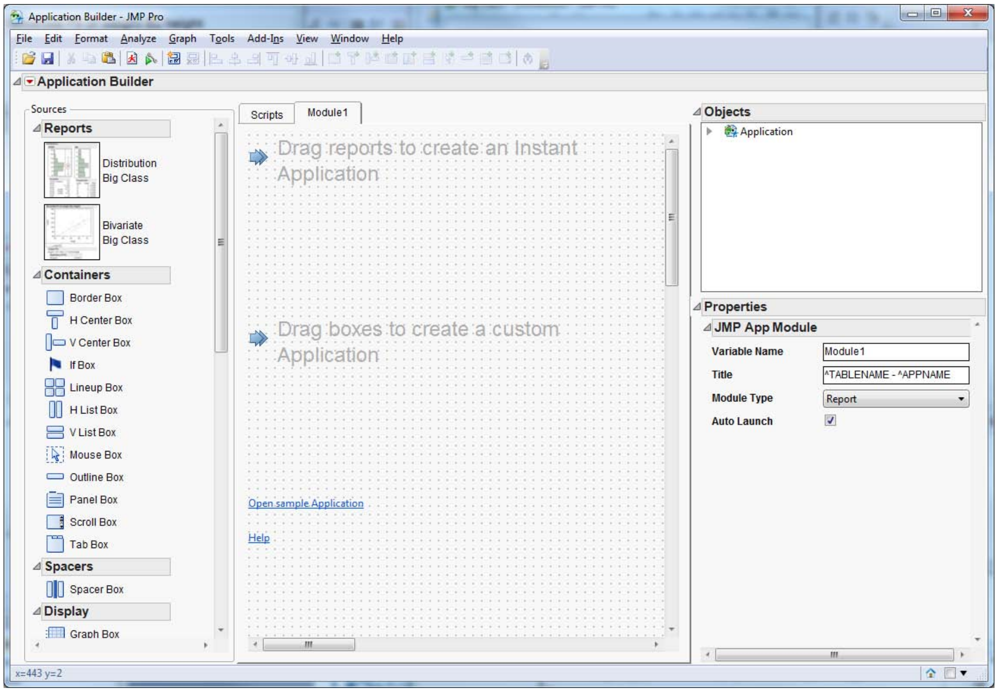

Run the Module Script.
Create the display and platform objects, if they were not already created through the script.

Figure 2: Execution Flow for a Module Instance

The Module is also a lightweight wrapper around the objects that it contains. Applications consisting primarily of JMP platforms might require no additional scripting, while the availability of the script provides a mechanism for customized processing and workflow.

BUILDING JMP APPLICATIONS

The Application Builder is a graphical environment for creating JMP Applications. Figure 3 shows the initial view of the Application Builder.



Drag reports to create an Instant Application

Drag boxes to create a custom Application

Variable Name	Module1
Title	^TABLENAME - ^APPNAME
Module Type	Report
Auto Launch	☑

Figure 3: Initial Application Builder View

The Application Builder consists of the following four main areas:

- The Sources panel on the left shows the graphical content that is available to add to the application. The sources include full JMP platforms that have already been created, as well as individual display boxes for building and arranging custom dialog boxes and reports.
- The workspace in the center of the screen is where the layout of the application window is defined. For multi-window applications, there is one tab for each different type of window, as well as a Scripts tab for defining the Application and Module scripts.
- The Objects panel at the upper right displays a hierarchical view of the Application, including each Module and display box. This tree provides an alternate method for selecting objects, which can be easier than clicking in the workspace depending on the type and size of the object.
- The Properties panel at the lower right displays the properties for the currently selected object(s). Properties can be edited according to their type, including strings, numbers, fonts, colors, list editors, option selection, Boolean toggles, and several custom editors for specific properties.

The toolbar at the top of the window provides quick access to many common features (such as copying and pasting objects) and Application Builder specific operations (such as aligning boxes and adding borders). Many of these operations can also be performed from the right-click menu in the workspace or object tree.

The initial workspace provides a link to the Application Builder documentation and sample applications. The sample applications demonstrate several application patterns and are a useful starting point for developing similar applications.

INTERFACE LAYOUT

To begin creating a GUI layout, double-click objects on the Sources panel or drag them onto the workspace, as illustrated in Figure 4.

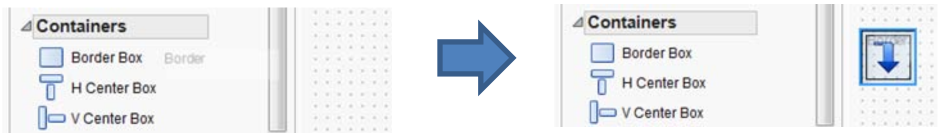


Figure 4: Creating Objects from the Source Panel

Objects in the Sources panel are organized by function. Open reports generated from a JMP data table appear under ***Reports***. The reports might contain a large amount of active functionality for setting parameters on the analysis, selecting and brushing data points, and more live interaction linked to the data table. ***Containers*** are display boxes that organize the display of one or more *child* boxes. ***Spacers*** are boxes that add padding between other boxes. The ***Display*** category includes boxes that show information, such as text, numbers, icons, and pictures. ***Tables*** include all of the elements for creating a table of results, including boxes for numbers and string columns. ***Buttons*** are the boxes that are used to perform an action or select items from a list. ***Input*** boxes allow the user to enter information. The ***Data Filter*** category includes boxes for applying local filtering to hide or exclude rows from a subset of the display.

Containers define the layout of a graphical display and allow the boxes to move with relation to one another when content is shown, hidden, or changes size. Drag an object into a container, or right-click the object and select **Add Container**. Different containers have different behavior, so during a drag operation a highlight will be shown to indicate where the object will be inserted when dropped. Figure 5 shows how to insert a new button before two existing buttons in a horizontal List Box.

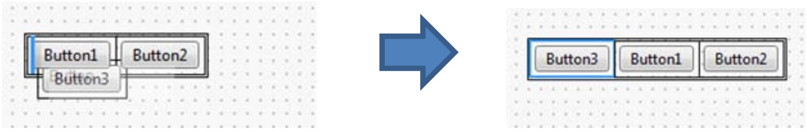


Figure 5: Inserting a New Object into a Horizontal List Box

In addition to top-down design, Application Builder also supports operations for bottom-up design. You might find that it is easier to experiment by placing your boxes manually, without containers. When you are ready to join boxes into a container for layout control, simply select the multiple boxes, right-click to select *Add Container*, and select the desired container. Figure 6 shows that this approach can result in the same final display, giving you the flexibility to choose a layout later in the development process.

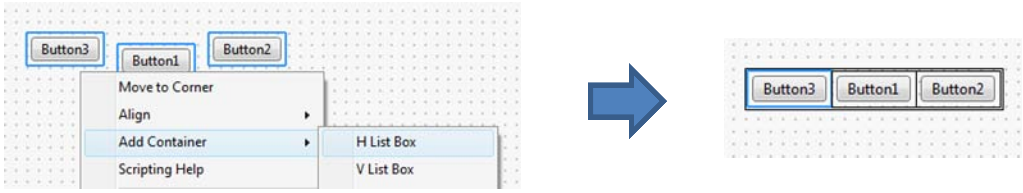


Figure 6: Adding a Container to Arrange Multiple Boxes

The *Add Container* command also lets you add a container around a box that already has a parent container. A complementary *Remove Container* command lets you remove the container of the selected box. The Add Container and Remove Container commands are also available from the toolbar ( ) and Objects panel.

While it is common to use containers to control the relative positions of boxes, you can also choose absolute positioning. By default, a grid is displayed in the background and box positions snap to this grid, making it easier to align objects by the left or top border. If you turn the snap option off or require other types of alignment, simply select the multiple boxes, right-click to select *Align*, and find the desired alignment option.

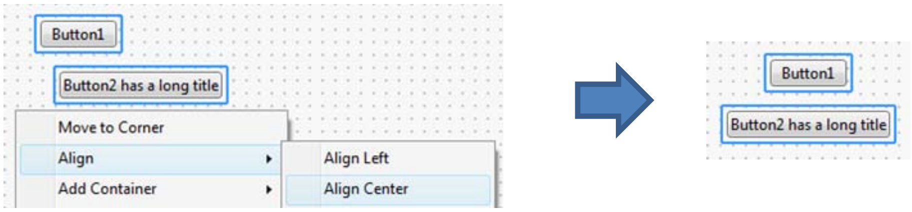


Figure 7: Aligning Display Boxes

The alignment options are also available from the toolbar ( ) as well as from the Object Tree. One advantage to using containers is that the box layout will automatically reconfigure based on the sizes of each object, which might change based on font, text, column names, or other factors.

Properties for the selected objects are displayed in the lower right corner of the Application Builder window. All objects share three properties: a *Variable Name* and *X / Y Positions*. The variable name is a JSL variable that is created in the namespace of the Module Instance. The variable can be referenced in the Module script to query or make changes to the object at run time. The X and Y positions determine the location of the object in the workspace, unless the object is managed by a container. The rest of the properties are object specific. For more information about the properties of a particular object, right-click and select *Scripting Help*. The Scripting Index opens to provide details about the selected object.

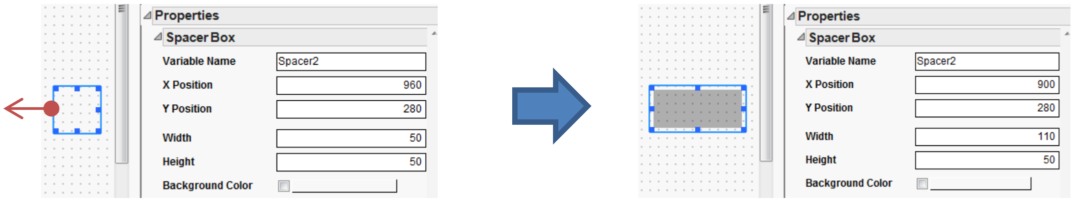
Each property has a unique editor based on the parameter type. Table 3 shows the editors for various properties.

Numeric	20	**String**	Text1
Boolean	☑	**Option**	Horizontal ▾
Color	☑ 🟦	**Script**	... CheckBox1Change
List	Item1 Item2	**Aggregate**	□ 0 □ 0 0 □ 0 □
Font	... Arial, Plain, 9pt		

Table 3: Properties and Editors

The aggregate editor in this example is used for several objects to provide a visual representation of properties controlling spacing on top, bottom, left, and right.

To position an object, edit the object properties or drag-and-drop the object in the workspace. Some objects also support visual editing of other properties using drag handles that appear when the object is selected, as shown in Figure 8.



Properties

Spacer Box

Variable Name	Spacer2
X Position	960
Y Position	280
Width	50
Height	50
Background Color	□

Properties

Spacer Box

Variable Name	Spacer2
X Position	900
Y Position	280
Width	110
Height	50
Background Color	□

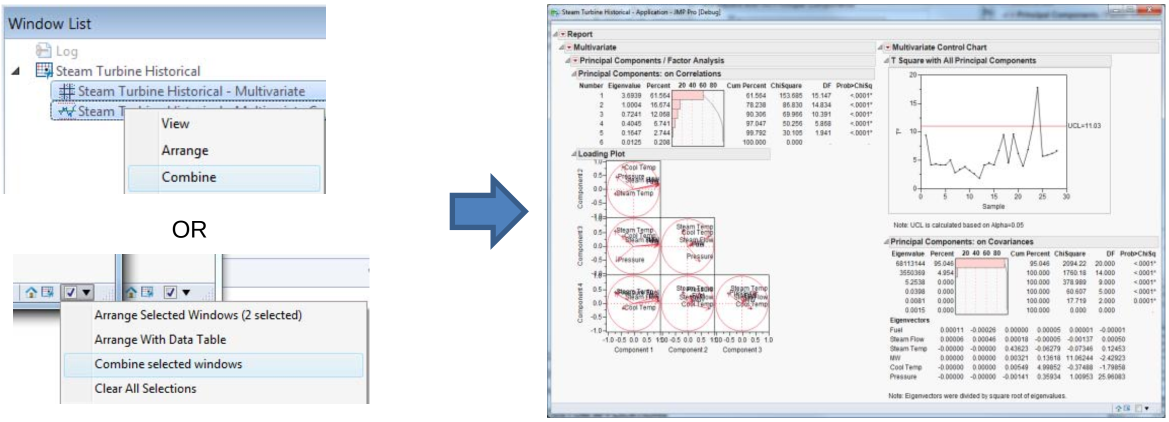
Figure 8: Editing an Object Using Drag Handles

Collections of boxes can be moved or copied by selecting the objects and using the *Cut* or *Copy* operation from the Edit menu, toolbar, or the right-click menu. When objects are copied, all of the properties are copied to the new objects, including any referenced scripts. The object names will be updated to ensure that they are unique, and any references to these names in the copied script will also be updated to match the new names. Copy and paste works across applications as well, making it easy to re-use part of an existing application in a new application.

The following sections provide more detail about the process of creating two classes of JMP Applications: Instant Applications and Custom Applications.

INSTANT APPLICATIONS

Instant Applications combine multiple analyses into one report. On the Windows platform this process is streamlined through integration into the Home window and the Report windows. Start by selecting multiple reports and select the *Combine* or *Combine selected windows* option. Application Builder creates a new window with the reports embedded, as shown in Figure 9.


Window List
Log
Steam Turbine Historical
Steam Turbine Historical - Multivariate
Steam T
View
Arrange
Combine
OR
Arrange Selected Windows (2 selected)
Arrange With Data Table
Combine selected windows
Clear All Selections
Steam Turbine Historical - Application - JMP Pro [Debug]
Report
Multivariate
Principal Components / Factor Analysis
Principal Components: on Correlations
Loading Plot
Component 1
Component 2
Component 3
Component 4
Cool Temp
Pressure
Steam Temp
Steam Flow
Multivariate Control Chart
T Square with All Principal Components
T²
Sample
UCL=11.03
Principal Components: on Covariances

Number	Eigenvalue	Percent	Cum Percent	ChiSquare	DF	Prob>ChiSq
1	3.6939	61.564	61.564	153.685	15.147	<.0001*
2	1.0004	16.674	78.238	86.830	14.834	<.0001*
3	0.7241	12.068	90.306	69.956	10.391	<.0001*
4	0.4045	6.741	97.047	50.256	5.868	<.0001*
5	0.1647	2.744	99.792	30.105	1.941	<.0001*
6	0.0125	0.208	100.000	0.000		

Note: UCL is calculated based on Alpha=0.05

Eigenvalue	Percent	Cum Percent	ChiSquare	DF	Prob>ChiSq
68113144	95.046	95.046	2094.22	20.000	<.0001*
3550369	4.954	100.000	1760.18	14.000	<.0001*
5.2538	0.000	100.000	378.989	9.000	<.0001*
0.0398	0.000	100.000	60.607	5.000	<.0001*
0.0081	0.000	100.000	17.719	2.000	0.0001*
0.0015	0.000	100.000	0.000	0.000	

Eigenvectors

Fuel	0.00011	-0.00026	0.00000	0.00005	0.00001	-0.00001
Steam Flow	0.00006	0.00046	0.00018	-0.00005	-0.00137	0.00050
Steam Temp	-0.00000	-0.00000	0.43623	-0.06279	-0.07346	0.12453
MW	0.00000	0.00000	0.00321	0.13618	11.06244	-2.42923
Cool Temp	-0.00000	0.00000	0.00549	4.99852	-0.37488	-1.79858
Pressure	-0.00000	-0.00000	-0.00141	0.35934	1.00953	25.96083

Note: Eigenvectors were divided by square root of eigenvalues.

Figure 9: Combining Multiple Reports to Create an Instant Application

The automated combine function will attempt to replicate the layout of the reports on your screen, in horizontal or vertical direction. For more control over layout, the red triangle menu for the application has an option to load the resulting application into Application Builder. All functions of Application Builder are available, including reordering of reports or creating new container boxes to rearrange the layout.

You can also combine reports into an Instant Application inside Application Builder. The first category in the Sources panel is *Reports*, which contains a thumbnail image of each current report window. These full reports can be treated just like the basic display boxes. Double-click or drag to create a new instance of the report in Application Builder. Because a report is the result of a live platform, it is not possible to rearrange the components of the report. The internal structure of live reports can be modified only through controls provided by the platform.

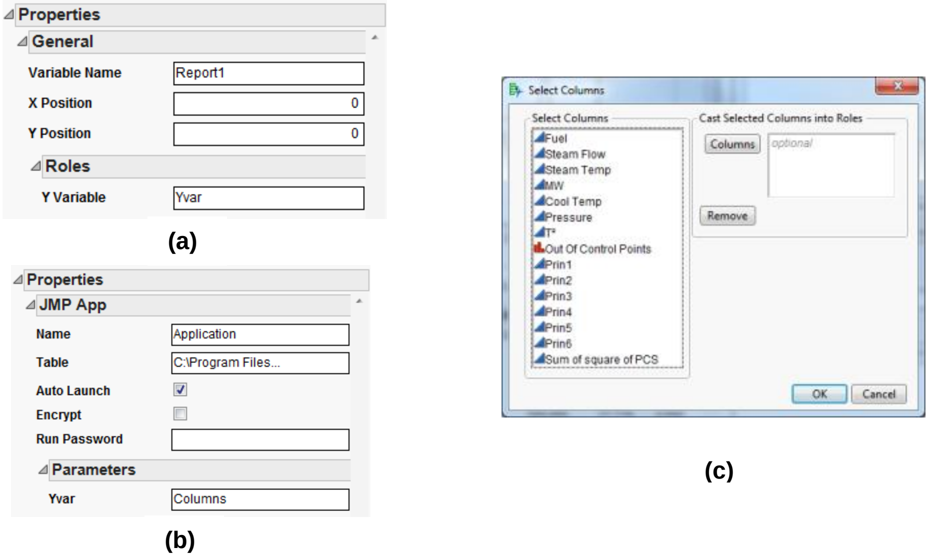
The default behavior of a report embedded in an application is to reproduce the analysis using the same columns that were used in the original analysis. This is very similar to saving the script for each platform and then combining the scripts together in a new window. This is a useful first step for creating applications, but often there are changes that you want to make at run time. Two additional features greatly expand the class of Instant Applications that are possible – all without writing a single line of JSL.

Parameterizing Instant Applications

Parameterized Instant Applications are combinations of reports in which the column roles have been assigned the name of an application parameter. At run time, you are prompted to assign values to the application parameters, which are then passed to the platforms prior to creating the reports. Application parameters can span multiple reports, so you can specify that two platforms should use the same Y columns by assigning them the same parameter name.

There are two steps to specifying the application parameters. The first step is to specify which platform roles you want to parameterize. When a report is selected, the property panel shows a list of the roles for the platform. For each role that you want to parameterize, enter a JSL variable name, as shown in Figure 10(a). When you are prompted for column roles at run time, the selected column(s) will be stored in this JSL variable.

The second step is optional. By default, Application Builder uses the name of the platform role when prompting for the application parameter. If the platform has a *Size* role, you will be prompted to select a *Size* column. You can change this label by selecting the Application object in the Objects panel. In the Properties panel for the Application object there is a list of all application parameters that have been created, as shown in Figure 10(b). Each role appears only once, even if it has been used multiple times for different platforms. The resulting run-time launch dialog box is shown in Figure 10(c).



Properties
General
Variable Name: Report1
X Position: 0
Y Position: 0
Roles
Y Variable: Yvar

(a)

Properties
JMP App
Name: Application
Table: C:\Program Files...
Auto Launch: ☑
Encrypt: ☐
Run Password:
Parameters
Yvar: Columns

(b)

Select Columns
Select Columns
Fuel
Steam Flow
Steam Temp
MW
Cool Temp
Pressure
T^2
Out Of Control Points
Prin1
Prin2
Prin3
Prin4
Prin5
Prin6
Sum of square of PCS
Cast Selected Columns into Roles
Columns: optional

(c)

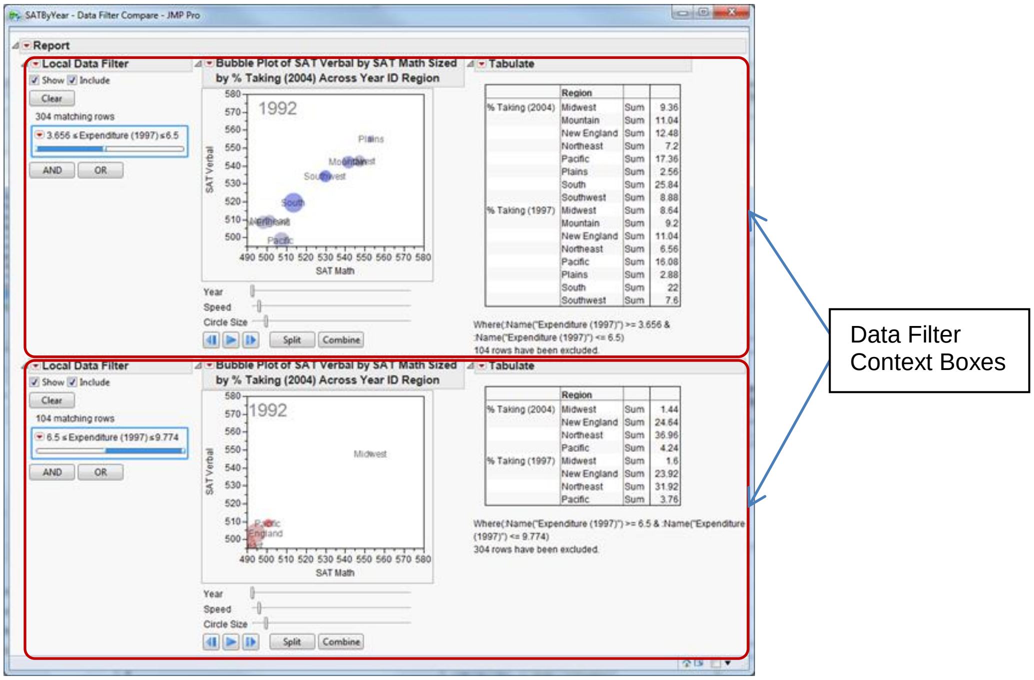
Figure 10: Parameterizing a Platform Role. (a) Select the roles to parameterize (b) Specify a label for the parameter (c) The Resulting Launch Dialog Box at Run Time

When you create parameters that are shared across multiple platform roles, Application Builder will keep track of known restrictions related to data type, modeling type, and number of parameters. These limits will be merged together and used in the combined launch dialog box to guide the user in selecting valid input. An error message will be displayed during parameterization if conflicts are detected between multiple roles that are being combined.

Embedded Data Filters

The local data filter is new to JMP 10. While the traditional data filter makes changes to the data table that are globally visible, the local data filter is integrated into platforms and provides for local control of the visibility and inclusion of rows. Local data filters are also supported from JSL and Application Builder, with a special container box called the Data Filter Context Box that defines the extent of the filter effect.

One advantage of using embedded filters for Instant Applications is that a single filter can be shared among multiple platforms, by arranging them within the same Data Filter Context Box. You can also include multiple filters, either by nesting the context boxes or using them to divide the window in multiple filter regions. Figure 11 shows the sample application *Data Filter Compare* with two embedded data filters, each shared between a Bubble Plot and Tabulate platform.



	Region		
% Taking (2004)	Midwest	Sum	9.36
	Mountain	Sum	11.04
	New England	Sum	12.48
	Northeast	Sum	7.2
	Pacific	Sum	17.36
	Plains	Sum	2.56
	South	Sum	25.84
	Southwest	Sum	8.88
% Taking (1997)	Midwest	Sum	8.64
	Mountain	Sum	9.2
	New England	Sum	11.04
	Northeast	Sum	6.56
	Pacific	Sum	16.08
	Plains	Sum	2.88
	South	Sum	22
	Southwest	Sum	7.6

Where(:Name("Expenditure (1997)") >= 3.656 &
:Name("Expenditure (1997)") <= 6.5)
104 rows have been excluded.

	Region		
% Taking (2004)	Midwest	Sum	1.44
	New England	Sum	24.64
	Northeast	Sum	36.96
	Pacific	Sum	4.24
% Taking (1997)	Midwest	Sum	1.6
	New England	Sum	23.92
	Northeast	Sum	31.92
	Pacific	Sum	3.76

Where(:Name("Expenditure (1997)") >= 6.5 & :Name("Expenditure (1997)") <= 9.774)
304 rows have been excluded.

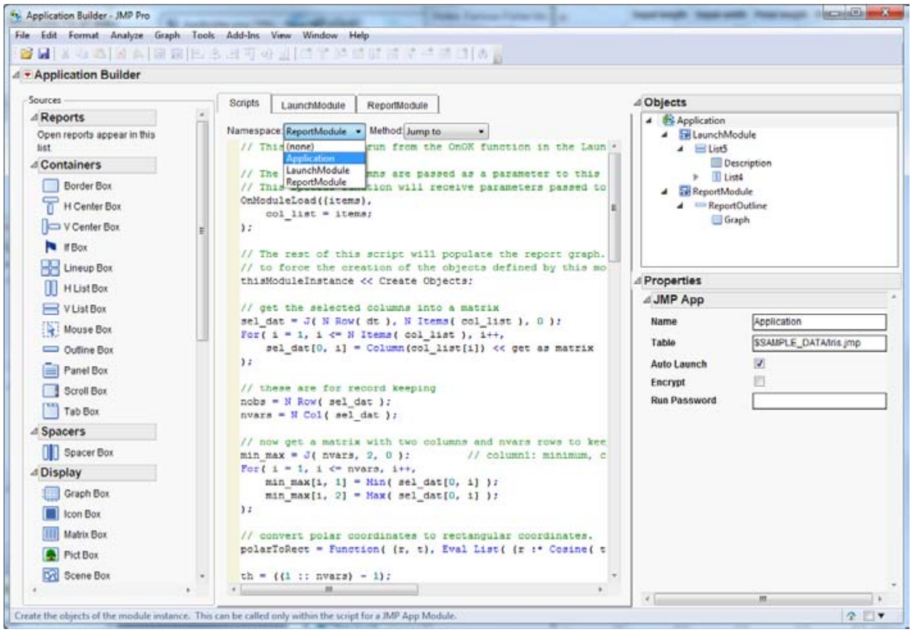
Data Filter Context Boxes

Figure 11: Embedded Local Data Filters Shared by Multiple Platforms

When arranged in a nested configuration, the results of the filters will aggregate from the outside-in. Typically you would filter on different columns in a hierarchical configuration. For example, in Figure 11 you might consider adding a filter around the entire display to filter all of the results by population.

CUSTOM APPLICATIONS

Custom applications integrate analyses from JMP with custom JSL scripts to perform additional analysis and produce new reports. The custom scripts have full access to JSL functionality, including local JSL functions and remote execution of SAS or R components. Application scripts are edited through the integrated Scripts tab in the workspace, as shown in Figure 12.



Namespace: ReportModule Method: Jump to

```
// This                run from the OnOK function in the Laun

// The            mns are passed as a parameter to this
// This                ion will receive parameters passed to
OnModuleLoad({items},
    col_list = items;
);

// The rest of this script will populate the report graph.
// to force the creation of the objects defined by this mo
thisModuleInstance << Create Objects:

// get the selected columns into a matrix
sel_dat = J( N Row( dt ), N Items( col_list ), 0 );
For( i = 1, i <= N Items( col_list ), i++,
    sel_dat[0, i] = Column(col_list[i]) << get as matrix
);

// these are for record keeping
nobs = N Row( sel_dat );
nvars = N Col( sel_dat );

// now get a matrix with two columns and nvars rows to kee
min_max = J( nvars, 2, 0 );          // column1: minimum, c
For( i = 1, i <= nvars, i++,
    min_max[i, 1] = Min( sel_dat[0, i] );
    min_max[i, 2] = Max( sel_dat[0, i] );
);

// convert polar coordinates to rectangular coordinates.
polarToRect = Function( {r, t}, Eval List( {r :* Cosine( t

th = ((1 :: nvars) - 1);
```

Name	Application
Table	$SAMPLE_DATA/Iris.jmp
Auto Launch	☑
Encrypt	☐
Run Password	

Create the objects of the module instance. This can be called only within the script for a JMP App Module.

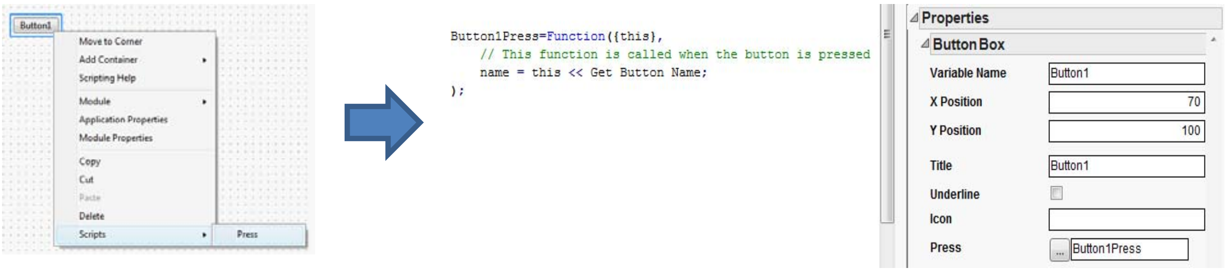
Figure 12: Selecting an Application Script to Edit

The namespace menu has entries for the Application as well as each Module defined in the application. A list of methods or functions provides a convenient way to navigate through large scripts.

In addition to the JSL variable names that have been specified through properties on the boxes and Modules, there are two special JSL variables that can be used in your scripts. The `thisApplication` variable refers to the Application object, and can be used from any Application script. The `thisModuleInstance` variable can be used from any Module script and refers to the running instance of the Module.

Scripting Objects

Custom applications will use more of the available display boxes for both input and display of information and graphics. Actions associated with display boxes are controlled through JSL scripts that are part of the properties associated with an object. For simple actions you might find it convenient to store the entire script in the property. For more complex actions, or actions that can be invoked through multiple means, you might prefer to define a function in the Module scope and invoke this function from the object property. To help you with these linked actions, you can right-click on an object and navigate to the *Scripts* menu. Each script property associated with the selected object is shown in the menu. As shown in Figure 13, selecting a script property from this list creates a template function for the action and associates the new function with the box. If you select a property that already has an associated function, the script editor will open to the given function.



```
Button1Press=Function({this},
    // This function is called when the button is pressed
    name = this << Get Button Name;
);
```

Variable Name: Button1
X Position: 70
Y Position: 100
Title: Button1
Underline: □
Icon:
Press: Button1Press

Figure 13: Adding a Script Action to an Object

The template functions give you a starting point for defining a function, including comments on the behavior of the function and a valid function signature with correct arguments. Each function will have a *this* argument, giving you a handle to the object that was activated. Using the *this* parameter will make it easier to share functions across multiple objects. Some objects might pass additional parameters, particularly in cases when the additional information cannot be queried from the object using scripting.

Module Flow Control

While Instant Applications have a relatively simple and fixed flow of control, custom Applications require more flexibility in order to implement different classes of applications. A JMP Application Module corresponds to a window that is created at run time. Each Module has an *Auto Launch* property that controls whether an instance of the Module is created automatically when the application is run. This is true by default and makes development of Instant Applications possible without requiring JSL scripting. But it is vital for the programmer to be able to take control of the flow in order to support a variety of application patterns. Some common patterns might include:

- *Launch / Report*: A launch window is presented to gather input from the user. On completion of the input, the launch dialog box is dismissed, computation is completed, and the associated report window appears.
- *Report generator*: A window appears, showing a variety of reports and parameters that are available. The user can selectively launch reports, each of which results in a new window. The same report might be launched multiple times with different parameters.

The Auto Launch property is the first tool in controlling the flow of the application, by controlling the initial view that the user sees. From this point on, the flow of the application is controlled through the JSL scripts defined by the application. In response to button presses or other actions, it might be necessary to create a new instance of an Application Module. This is implemented through a message to the Module object:

```
Module1 << Create Instance(param);
```

Each Module has a JSL variable name that is automatically visible from any script in the Application. The parameter list gives you the ability to pass values to the module that is used in the computation or display. This could include column names, thresholds, or other input required by the Module to complete its analysis and report. The parameters are received by the Module using a special function defined in the Module script:

```
        OnModuleLoad({param}, <body of function>)
```

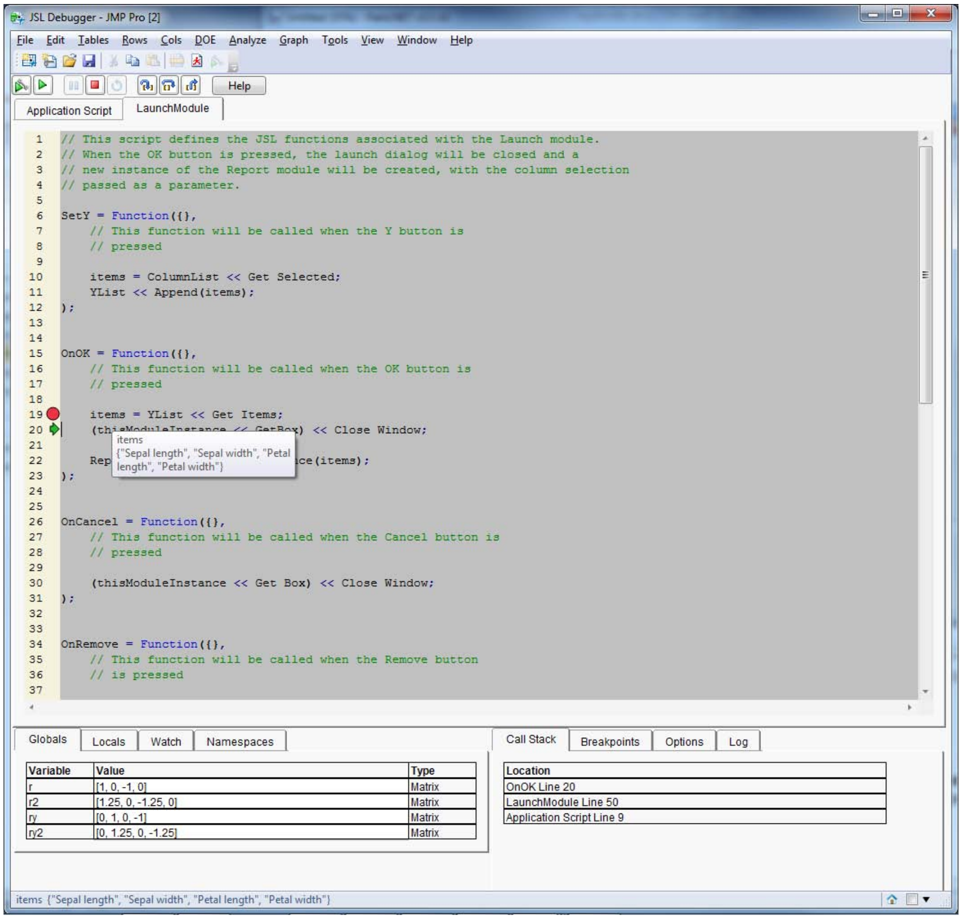
This function can appear anywhere in the body of the Module script. Generally it is useful to process the input early in the script, but there might be some initialization that is required earlier. Another useful statement that appears in the template for the Module script is:

```
        thisModuleInstance << Create Objects;
```

This statement allows you to control when the box objects and reports are created. It is not required that you include this statement in the Module script, but including it will allow you to send messages to the created objects or set properties on the object at the time that the Module Instance is created. This is useful when the properties on the object are not known when the Application is being developed, for example if you want to use the name of the data table in a text field.

DEBUGGING APPLICATIONS

New with JMP 10 is a JSL debugger environment with flow control, breakpoints, and variable inspection. The JSL debugger can be used on stand-alone scripts and is also integrated into the Application Builder environment. You can begin a debugging session from the Edit menu, the toolbar, or by selecting the option *Debug Application* from the red triangle menu of Application Builder. As each Application or Module Instance is run, the associated script is loaded into the debugger. You can set breakpoints, step into or over statements, and inspect variables as they are modified. Figure 14 shows an example session with a breakpoint set in the callback function for a launch dialog box.


JSL Debugger - JMP Pro [2]
File Edit Tables Rows Cols DOE Analyze Graph Tools View Window Help
Help
Application Script
LaunchModule
Globals
Locals
Watch
Namespaces
Call Stack
Breakpoints
Options
Log

```
// This script defines the JSL functions associated with the Launch module.
// When the OK button is pressed, the launch dialog will be closed and a
// new instance of the Report module will be created, with the column selection
// passed as a parameter.

SetY = Function({},
    // This function will be called when the Y button is
    // pressed

    items = ColumnList << Get Selected;
    YList << Append(items);
);


OnOK = Function({},
    // This function will be called when the OK button is
    // pressed

    items = YList << Get Items;
    (thisModuleInstance << GetBox) << Close Window;

    Rep                                   ce(items);
);


OnCancel = Function({},
    // This function will be called when the Cancel button is
    // pressed

    (thisModuleInstance << Get Box) << Close Window;
);


OnRemove = Function({},
    // This function will be called when the Remove button
    // is pressed

```

items
{"Sepal length", "Sepal width", "Petal length", "Petal width"}

Variable	Value	Type
r	[1, 0, -1, 0]	Matrix
r2	[1.25, 0, -1.25, 0]	Matrix
ry	[0, 1, 0, -1]	Matrix
ry2	[0, 1.25, 0, -1.25]	Matrix

Location
OnOK Line 20
LaunchModule Line 50
Application Script Line 9

items {"Sepal length", "Sepal width", "Petal length", "Petal width"}

Figure 14: Debugging an Application

The debugger will automatically terminate when it detects that the application has completed.

DEPLOYING APPLICATIONS

Once you build an Application, you will want to distribute and maintain them. There are several mechanisms to do this, depending on your needs.

There are two new file types associated with JMP Applications. The *.jmpappsource* file is the development version of the application. When opened through the File menu or through a JSL `Open()` command, the Application Builder environment initializes and loads the contents to continue editing the application. The *.jmpapp* file is the executable version of the application. When this file is opened, the Application object is loaded and run. Both files are text based and contain similar content. The .jmpappsource file is never encrypted, while the *.jmpapp* file will be encrypted if the property has been set on the Application object.

Two options are supported that are familiar to users of JMP platforms. *Save Script to Data Table* and *Save Script to Journal* are available from the red triangle menu of the Application Builder. As shown in Figure 15, saving an Application script to the Data Table embeds the application as a property on the table. This is a full copy of the Application, and not a reference to a separate file. This allows you to communicate results by sharing only a single file, with all of your analysis included in the file.

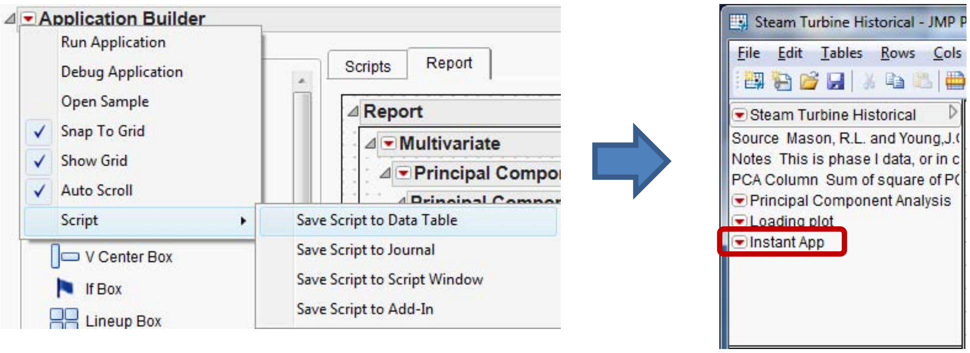
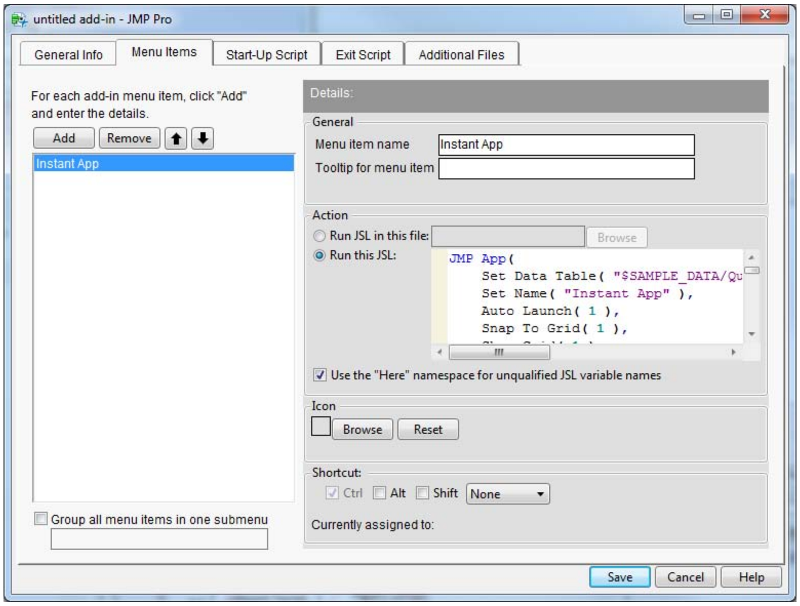


Figure 15: Saving an Application Script to the Data Table

The *Save Script to Journal* option embeds the application in the current journal, or creates a new one if one does not exist. The embedded application appears as a script connected to a button, allowing you to launch the application from the journal. This is often used as a technique to create an outline for a presentation.

Launching an application from an add-in is useful for both personal use and for distribution of applications to other users. Add-Ins are JMP extensions that are recognized and added to the local JMP installation when they are opened. An add-in can create one or more buttons in the JMP menu to create a persistent way to launch a custom script, so the user will not have to remember where they saved the application file. When you select *Save as Add-In* from the red triangle menu of Application Builder, the Add-In Builder opens to let you customize the add-in. Figure 16 shows one of the tabs of the Add-In Builder as invoked from Application Builder.



For each add-in menu item, click "Add" and enter the details.

Instant App

☐ Group all menu items in one submenu

General

Menu item name: Instant App

Tooltip for menu item:

Action

○ Run JSL in this file:

● Run this JSL:

```
JMP App(
    Set Data Table( "$SAMPLE_DATA/Qu
    Set Name( "Instant App" ),
    Auto Launch( 1 ),
    Snap To Grid( 1 ),
```

☑ Use the "Here" namespace for unqualified JSL variable names

Icon

Shortcut:

☑ Ctrl ☐ Alt ☐ Shift None

Currently assigned to:

Figure 16: Saving an Application to an Add-In

In the Add-In, the name of the menu item has been initialized to the name of the application, and the Application script is used as the script to run. Saving this document will result in a *.jmpaddin* file that can be distributed to other users. When you open an add-in JMP prompts you to install it, after which it can be run from the specified menu. Add-Ins have advantages over JSL scripts and JMP Application files when communicating and distributing results because they integrate into both the installation and the menu system of JMP.

CONCLUSION

The JMP Application Builder provides a framework for application development in JMP by using intuitive drag-and-drop construction techniques for GUI layout. The property editor relieves the programmer from having to remember all object properties and valid parameters. For the non-programmer, Instant Applications enable you to create custom combinations of JMP platforms. By parameterizing the Instant Application, you can create general-purpose applications that can be used on data from multiple sources and with multiple queries. Advanced JSL developers will appreciate the automated management of namespaces to avoid unintended interaction between scripts. The increased focus on scripting for analysis rather than scripting for GUI design allows the advanced programmer to concentrate more on the application, and less on how it is delivered. Future development will continue to integrate additional GUI elements, support more advanced Instant Applications, and add more scripting help for the advanced custom application developer.

REFERENCES

Hill, Eric. 2011. “JMP® 9 Add-Ins: Taking Visualization of SAS® Data to New Heights”. Proceedings of the SAS Global Forum 2011 Conference. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings11/TOC.html.

Miclaus, Kelci. 2011. “JMP® as an Analytic Hub: Using JMP to Build Custom Applications via SAS® and R”. Proceedings of the SAS Global Forum 2011 Conference. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings11/TOC.html.

Watson, Wayne. 2011. “Introducing SAS® Structural Equation Modeling for JMP®: A New User Interface that Brings the Power of SAS/STAT® Software to JMP Software”. Proceedings of the SAS Global Forum 2011 Conference. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings11/TOC.html.

ACKNOWLEDGMENTS

I would like to thank Craige Hales for his initial work and continued input on Application Builder. Xan Gregg, Joseph Morgan, and Jeff Perkinson provided feedback and suggestions throughout the development. Rosemary Lucas, Hui Di, Melanie Drake, Kyoko Keener, Sheila Loring, Liza Lucas and Nala Sun provided valuable early testing and feedback. Chung-Wei Ng developed the Local Data Filter, Dave Barbour developed the Add-In Builder, and Jeff Polzin developed the JSL Debugger. Their help integrating these new features with Application Builder is greatly appreciated.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Daniel Schikore
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
(919) 531-8670
Dan.Schikore@jmp.com
http://www.jmp.com