
Automating JMP® using C#

Ryan Gilmore, SAS Technical Support

Table of Contents

Automating JMP® Using C#.....	1
How to Create a Data Table	1
How to Run an Analysis.....	3
How to Display and Select Analysis Results.....	4
How to Save the Results	5
Automation and JSL	5

Summary	7
----------------------	----------

Automating JMP® Using C#

The purpose of this paper is to provide information on getting started with automating JMP® 6 using C# .NET. This document discusses how to automate the more common tasks involved in analyzing data such as how to create a data table, run an analysis, display options within the analysis, select portions of the analysis report, and save results to a file. The final section describes the interface between automation and the JMP Scripting Language (JSL).

This paper does not discuss how to setup a project or create forms. The code provided was written using Microsoft Visual Studio .NET 2003.

How to Create a Data Table

There are several methods for creating a JMP data table.

- Open a file with a known extension.
- Open a table within a database.
- Open a text file.
- Create the table from scratch.

To open a file with a known extension, use the **OpenDocument** method of the **Application** object. JMP will then open the file based on the extension. A list of known extensions can be found by selecting **Open** from the **File** menu then viewing the “Files of type” field in the “Open Data File” dialog. The code below shows how to open an Excel file,

```
JMP.Application myJMP;  
JMP.Document doc;  
  
myJMP = new JMP.Application();  
doc = myJMP.OpenDocument("C:\\MyData\\BigClass.xls");
```

To open a file from a database such as Oracle, use the **AUTODB** object. For example,

```
JMP.Application myJMP;  
JMP.AUTODB db;  
JMP.DataTable dt;  
  
myJMP = new JMP.Application();  
  
// Create a new AUTODB object.  
db = myJMP.NewDatabaseObject();  
  
// Connect to the database.  
db.Connect("DSN=Oracle;USR=yyyyy;PWD=xxxxx");  
  
// Retrieve the table in the database.  
// The returned result is a reference to the JMP  
// data table created.  
  
dt = db.OpenTable("bigclass");  
// Disconnect from the database.  
db.Disconnect();
```

The information required to connect to the database in the **Connect** method can be obtained by connecting to the database manually through JMP then viewing the connection string in the Log window. JMP uses ODBC to communicate with other databases. For more information about JMP and ODBC, refer to the white paper entitled, *Connections: JMP and ODBC*.

To open a text file and specify the import characteristics, use the **TextImport** object as follows,

```
JMP.Application myJMP;
JMP.Document doc;
JMP.TextImport ti;

myJMP = new JMP.Application();

// Create the TextImport object
// The first parameter is the name of the file.
// The second parameter is the number of columns in the file.
ti = myJMP.CreateTextImportObject("C:\\MyData\\BigClass.txt", 5);

// Specify the import characteristics
ti.SetEndOfFieldOptions (JMP.jmpTIEndOfFieldConstants.tiTAB);
ti.SetEndOfLineOptions (JMP.jmpTIEndOfLineConstants.tiCRLF);
ti.FirstLineIsData(true);

// Import the text file. The returned result is a
// reference to a JMP document, not the data table.
doc = ti.OpenFile();
```

To create a data table from scratch, follow these general steps,

```
JMP.Application myJMP;
JMP.DataTable dt;

myJMP = new JMP.Application();

// Create an empty data table.
dt = myJMP.NewDataTable( "data_table_name" );

// Create a numeric column.
dt.NewColumn( "column_name",
             JMP.colDataTypeConstants.dtTypeNumeric,
             JMP.colModelTypeConstants.colModelTypeContinuous,
             8 );

// Create a character column of length 12.
dt.NewColumn( "column_name",
             JMP.colDataTypeConstants.dtTypeCharacter,
             JMP.colModelTypeConstants.colModelTypeNominal,
             12 );

// Add 100 rows to the data table.
short nRowsToAdd = 100;
int afterRowNumber = 1;
dt.AddRows( nRowsToAdd, afterRowNumber );

// Depending on the structure of the data being used to
```

```

// populate the data table, you can loop through each
// column and use the SetDataVector method of the column.

for(c = 1; c <= dt.NumberColumns; c++) {
    dt.GetColumnByIndex(c).SetDataVector( arrayRef );
}

// OR

// Use a loop to iterate over the rows and columns
// and use the SetCellVal method of the column.

JMP.Column col;
for(c = 1; c <= dt.NumberColumns; c++) {
    col = dt.GetColumnByIndex(c);
    for(r = 1; r <= dt.NumberRows; r++) {
        col.SetCellVal(r, "value ");
    }
}

```

When creating a data table with more than 20 rows, **SetDataVector** is recommended since it is faster than iterating with loops.

How to Run an Analysis

The steps required to run an analysis via automation are similar to those taken when performing the analysis manually within JMP. The basic tasks taken when running an analysis manually are:

1. Display the appropriate analysis dialog.
2. Cast the columns into roles.
3. Click the OK button.

The equivalent of displaying the analysis dialog in automation is to use the **Create** method of the **Document** object. The Create method does not display the dialog but rather returns a reference to the analysis object. For example, to start a Distribution analysis, use **CreateDistribution** as shown,

```

JMP.Application myJMP;
JMP.Distribution dist;
JMP.Document doc;

myJMP = new JMP.Application();
doc = myJMP.OpenDocument("C:\\MyData\\BigClass.xls");

dist = doc.CreateDistribution();

```

The next step, when doing this by hand, is to cast the columns into the appropriate roles within the launch dialog. The **Launch** methods of the **Platform** object accomplish this task in automation.

```
// Adds Height to the analysis
dist.LaunchAddColumn("height");
```

The **Launch** method is the equivalent of clicking the OK button within the dialog,

```
dist.Launch();
```

How to Display and Select Analysis Results

Display an option

To display an option manually in JMP, a selection is made from the pop-up menu after clicking the red triangle. For most platforms, each item within a pop-up menu has a corresponding automation method.

For example, with the Distribution platform, to display the count axis, use

```
dist.CountAxis(true);
```

or to change the layout of the distribution use

```
dist.HorizontalLayout(true);
```

Access Display Boxes

There are several methods available for obtaining references to various parts of the report. To begin the traversal of the report, start with either **GetGraphicItemByName** or **GetGraphicItemByType**. The **GetGraphicItemByName** is best used for referencing an outline box. For most other display boxes, **GetGraphicItemByType** is recommended.

Once the initial reference is obtained, use **GetSubgraphicItemByName** or **GetSubgraphicItemByType**. These will allow for nested traversal rather than having to know the exact position of the display box within the report.

The following example shows how to make a table from the Analysis of Variance report after running a bivariate analysis.

```
JMP.Document doc;
JMP.Bivariate biv;

myJMP = new JMP.Application();
doc = myJMP.OpenDocument("c:\\MyData\\big class.jmp");

biv = doc.CreateBivariate();
biv.LaunchAddX("weight");
biv.LaunchAddY("height");
biv.Launch();

// Fit a line once the analysis has been displayed.
JMP.Fit fLine;
fLine = biv.FitLine();

// Obtain a reference to the top item in the report
// which is the first outline box.
Int hOutlineBox = biv.GetGraphicItemByType("outline box", 1);
```

```

// Make a data table from the first table box
// under the outline box titled "Analysis of Variance".
//
// The JSL equivalent would be
// report(biv)["Analysis of Variance"][table box(1)]
// where biv is the bivariate reference.
biv.TableBoxMakeDataTable(
    biv.GetSubgraphicItemByType(
        biv.GetSubgraphicItemByName(hOutlineBox,
            "Analysis of Variance"),
        "table box", 1)
);

```

For more information about display boxes, refer to the JMP Scripting Guide.

How to Save the Results

There are several choices available when saving the analysis.

To save the entire analysis window as a picture, use **SaveGraphicOutputAs**.

To save a particular graph or report, use **SaveGraphicItem**.

To save the analysis as a JMP journal, use **SaveJournalAs**.

The following code snippet shows how to save the same distribution analysis using the methods mentioned.

```

dist.SaveJournalAs("c:\\temp\\dist.jrn");

dist.SaveGraphicOutputAs("c:\\temp\\dist.jpg",
    JMP.jmpGraphicsFormats.jmpJPEG);

// SaveGraphicItem requires a handle to a display box so
// get the handle to the first outline box in the report.
int hDisplayBox = dist.GetGraphicItemByType("outline box", 1);
dist.SaveGraphicItem(hDisplayBox,
    "c:\\temp\\dist.png",
    JMP.jmpGraphicsFormats.jmpPNG);

```

Automation and JSL

While most tasks can be performed using the automation objects and methods provided, not all can. For those tasks, it may be worthwhile to see if they can be accomplished using JSL.

The automation interface provides three methods for submitting JSL code,

1. RunCommand
2. RunJSLFile
3. JSLFunction

The **RunCommand** and **RunJSLFile** methods are accessed through the Application object. The RunCommand is used to execute a few lines of JSL code. The RunJSLFile method is used to execute a JSL file stored on disk.

The **JSLFunction** method is accessed through the data table object. This method is used to submit a few lines of JSL code.

The difference between RunCommand and JSLFunction is that the JSLFunction method gives a return value which is the result of the last JSL statement submitted.

The automation methods for determining if a script was successful are **HasRunCommandErrorString** and **HasJSLFunctionErrorString**. **GetRunCommandErrorString** and **GetJSLFunctionErrorString** return descriptive text about the error.

Another useful method when interfacing with JSL is **GetJSLValue**. The GetJSLValue method can be used to extract a value that has been computed within JMP using JSL.

The following code shows how these methods could be used,

```
// Obtain the 97.5th percentile from the Quantiles
// report in the distribution analysis using JSL.
// The value is stored in the JSL global variable, q975.
myJMP.RunCommand("dist = distribution[1];" +
    "dist << quantiles(1);" +
    "q975 = report(dist)[\"quantiles\"] [table box(1)][3][3];");

// Check to see if there were any errors. If so,
// report them to the user and return from the routine.
if (myJMP.HasRunCommandErrorString()) {
    MessageBox.Show( myJMP.GetRunCommandErrorString() );
    return;
}

// Retrieve the value of the q975.
string q975 = myJMP.GetJSLValue("q975").ToString();
```

Summary

The automation interface to JMP contains a powerful set of objects and methods for manipulating JMP. This paper presents a subset, to help get started with the more common tasks of analyzing data. It begins with showing how to create a JMP data table using one of several methods then moves to creating, modifying, and saving the analysis. The final section describes the methods to use when needing to interact with JSL.



JMP Sales
SAS Campus Drive
Cary, NC 27513
US: 877-59-GOJMP
International: (1) 919-677-8000
Fax: (1) 919-677-4444
jmpsales@jmp.com
www.jmp.com

JMP Japan
Inui Bldg Kachidoki
1-13-1 Kachidoki
Chuo-ku Tokyo 104-0054 Japan
Tel: (81) 3 3533 3887
Fax: (81) 3 3533 1600
jmpjapan@sas.com
www.jmp.com/japan