Version 17

Automation Reference

"The real voyage of discovery consists not in seeking new landscapes, but in having new eyes."

Marcel Proust

JMP Statistical Discovery LLC
SAS Campus Drive
Cary, North Carolina 27513-2414

17.1

The correct bibliographic citation for this manual is as follows: JMP Statistical Discovery LLC 2022–2023. *JMP® 17 Automation Reference*. Cary, NC: JMP Statistical Discovery LLC

JMP® 17 Automation Reference

JMP Statistical Discovery LLC, SAS Campus Drive, Cary, North Carolina 27513-2414.

October 2022

March 2023

Contents

Document Object 98

Properties 98

Methods 98

Distribution Object Methods 119

DOE Object Methods 122

Running JMP Through External Applications

Most of JMP can be driven through OLE automation.

- "Automating JMP through Visual Basic" on page 36 introduces how to automate JMP through Visual Basic.
- "Automating JMP through Visual C++" on page 45 introduces automation using Visual C++ with MFC.
- "Automating JMP in Visual C#" on page 47 describes how to automate JMP through C#.
- "Application Object Reference for Automating JMP on Windows" on page 49 contains details for the methods and properties that JMP exposes to automation clients like Visual Basic and Visual C++.

The JMP/17/Samples/Automation folder contains several example Visual Basic .Net, and Visual C# .Net programs that automate features in JMP. The Visual Basic programs require Visual Studio 2013 or later.

Automating JMP through Visual Basic

Starting a JMP Application

The first step in automating JMP is to start it up. However, it's important to look at the resources available to help you with the JMP methods and properties. JMP provides a type library that allows automation controllers like Visual Basic (VB) to display a list of the methods and properties that JMP exposes, along with parameters that the methods require. This library is called JMP.TLB.

There are two steps to make the JMP type library available to VB.

1. Select **Project > References** in VB. A list of applications that are known to VB appears. If JMP is not in that list, select **Browse**. A file window asks you to locate a .tlb (Type library) file. Find the icon for the JMP type library in the JMP directory. Select this library and click **OK**.
2. Open the object browser by selecting **View > Object Browser** in VB. Select JMP from the drop down list box.

Now you can see the JMP automation classes and constants. You can now select a class, and the methods available to that class appear in the right list box for the object browser. If you select a method, a short helper string appears at the bottom of the window. This string lists the parameters for the method. Constants are used when methods require a restricted set of parameters, typically denoting a specific action.

Now that you have access to the type library information, write the necessary code to instantiate JMP. This is done with `CreateObject`. In global declarations for the VB project, create a variable of type *`JMP.Application`*. This is done as:

```
Dim MyJMP As JMP.Application
```

Now dimension some other variables. Good examples are `DataTable`, `Distrib`, `Oneway`, and `JMPDoc`. These are specified with `JMP.DataTable`, `JMP.Distribution`, `JMP.Oneway`, and `JMP.Document` respectively.

To create a JMP session, make it visible, and load a data table, add the following code to your VB script.

```
Dim JMPDoc As JMP.Document
Set MyJMP = CreateObject("JMP.Application")
MyJMP.Visible = True
Set JMPDoc = MyJMP.OpenDocument("C:\Program
Files\SAS\JMP\17\Samples\Data\Big Class.jmp")
```

The `Dim` statement indicates the type of variable. This declaration should go in the general declarations section of your VB project, though. If you do not do this, the JMP objects are destroyed when the variable goes out of scope at the end of the procedure.

JMP comes up invisible by default, as required by automation guidelines. Therefore, one of your first moves should be to make it visible, as shown in the above code.

Opening Multiple Instances of JMP

By default, JMP runs as a multiple-use automation server. This means that the first request to start JMP creates a new instance of the application. All subsequent client requests use the same instance.

JMP can also run as a single-use automation server. This means a new copy of JMP is created for each new request. To enable JMP as a single-user server, you need to modify a registry key.

If JMP is running with the same bit architecture (or, bitness) as the operating system, change the value of the following registry key to “Single” rather than “Multiple”.

```
HKEY_CLASSES_ROOT\CLSID\{97BCFCC0-7822-11CF-9E68-
00020AF24E9FE}\ServerUse
```

If JMP is 32-bit but the operating system is 64-bit, change the value of the following registry key to “Single” rather than “Multiple”.

```
HKEY_CLASSES_ROOT\Wow6432Node\CLSID\{97BCFCC0-7822-11CF-9E68-
00020AF24E9FE}\ServerUse
```

Warning: Always back up your registry before you make any registry changes. For assistance, see Windows Help, Microsoft documentation, or the Microsoft Windows web site. SAS is not responsible when you edit the Windows registry. Changes in the Windows registry can render your system unusable and would require you to reinstall the operating system.

Launching an Analysis

Now that you have a data table open, you can launch an analysis and manipulate it. Each analysis must first be created. Then, the required parameters for the analysis must be specified. Optional settings can also be specified. Then the analysis is launched. Additional option processing can then be done on the analysis object after the launch.

```
Dim Oneway As JMP.Oneway
Set Oneway = JMPDoc.CreateOneway
```

```
Oneway.LaunchAddY ("Height")
Oneway.LaunchAddX ("Age")
'Set an option before the launch
Oneway.Quantiles (True)
'Create the initial analysis output
Oneway.Launch
Oneway.MeansAnovaT (True)
Oneway.MeansStdDev (True)
Oneway.UnequalVariances (True)
Oneway.NormalQuantilePlot (True)
Oneway.SetAlpha (0.05)
Oneway.Save (oscCentered)
Oneway.Save (oscStandardized)
Oneway.CompareMeans occAllPairs, True
Oneway.CompareMeans occEachPair, True
```

The first step is to create the analysis object, which is done by calling the `CreateOneway` method of the document class. Next, X and Y columns are selected, and then `Launch` is called to create the actual One-way analysis. Each analysis platform has a distinct creation method, which you can view under the `Document` object in the object browser. In many cases, it is possible to specify options before the `Launch` of the object, so the analysis output uses the options that are already set. In this example, most option processing is done after the launch of the analysis, which shows the options popup in the display. As you can see, most methods are a simple setting of options, like you might do from a menu. `SetAlpha` takes a parameter, since you do not want to open a window for interaction during automation. `CompareMeans` takes two parameters, one for the type of comparison and one for the toggle to indicate on or off. The `Save` method takes a predefined constant (viewable in the object browser) that tells the Oneway analysis what to save.

Most analysis methods work this way, although some like Bivariate produce additional objects when methods are called. An example is:

```
Set Fit = Bivar.FitLine
Fit.ConfidenceFit (True)
Fit.ConfidenceIndividual (True)
```

Here, the `FitLine` method produces an object of type Fit. This object has methods and properties of its own, which can be manipulated. Remember, the new object created by `FitLine` can be manipulated only while its variable is in scope.

If a method produces an object that can also be automated, the object browser indicates this. For `FitLine`, the object browser specifies that the return type is `As Fit`.

Since this is not a predefined type like short or BSTR, you can probably guess that this is an object. If you look farther down the object browser, you see `Fit` as an object type. This confirms that an object is produced, and also gives you the methods that `Fit` supports.

Creating and Populating a Data Table

New data tables can be created with the (appropriately named) `NewDataTable` method of the `Application` object. A filename is assigned at creation time. This method returns a column object,

which must be retained as long as you want to add rows. By default, 20 rows are created. The `SetCellVal` method can be used to populate individual cells, and `AddRows` can be used to add rows as needed. Here is an example:

```
Dim Col As Object
Set DT = JMP.NewDataTable("C:\test.jmp")
Set Col = DT.NewColumn("Col1", dtTypeNumeric, 0, 8)
DT.Visible = True

'You must add rows before populating the table with data
DT.AddRows 20,0

'Set Cell values to increments of 1.5
For i = 1 To 10
      Col.SetCellVal i, i * 1.5
Next i
DT.Visible = False
For i = 11 To 20
      Col.SetCellVal i, i * 1.5
Next i
DT.Visible = True

'This adds 5 rows to the beginning of the table
DT.AddRows 5, 0
'This adds 5 rows after row 2
DT.AddRows 5, 2

'Now save the data table using the previously specified filename
DT.Document.Save

'If you wanted to create a subset of the table, with only rows 1-3
'you could do the following
'Note: you could also create subsets using specific columns by adding
the
'columns to a list using the AddToSubList member function of Datatable
Dim NewDT As JMP.Datatable
Dim DTDoc As JMP.Document
DT.SelectRows 1,3
Set NewDT = DT.Subset

'Now save the new table
Set DTDoc = NewDT.Document
DTDoc.SaveAs("C:\MySubset.jmp")
```

Example Programs

The JMP/17/Samples/Automation folder contains several example Visual Basic .Net, Visual C# .Net, and Visual C++ .Net programs that automate features in JMP. The Visual Basic programs require Visual Studio 2005 or later.

The ANALYSIS example program shows simple automation cases for almost all of the JMP platforms. The example code tests the features of a platform, but it does not pretend to do meaningful statistical analyses. Its purpose is for teaching automation coding. It is recommended that you make the JMP type library visible to the VB project. The first section of this document describes this process, which lets you see the methods and properties exposed by the automation platforms within JMP.

Likewise, the DATATAB example shows how to exercise the methods available for data table automation. No attempt is made to produce meaningful output.

The TIMPORT program shows the steps necessary to get a text file imported into JMP as a data table. Once this has been done, the data table can be manipulated just like the example in DATATAB, and analyses can be performed on the data just like in the ANALYSIS program.

The ODBCDemo program shows a simple example of importing a dBase file into JMP using ODBC access.

The WordDemo program shows the commands necessary to take a graphic section from a JMP report, copy it to the clipboard, and then insert it into a Microsoft Word document.

The FitModel and DOE examples show operators that are specific to those areas of JMP, and whose platform operator differs slightly from other platforms.

The sample code for all five example programs assumes the data files reside in the default SAMPLE DATA directory. If you move your sample data files, you need to change the path information in the VB samples.

If there are differences between this document's examples of Visual Basic code and that in the sample programs, preference should be given to the sample program code.

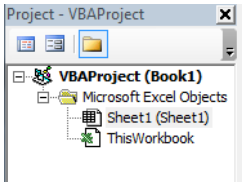
Automating JMP From Microsoft Excel 2013

This example automates JMP using a macro within a Microsoft Excel 2013 worksheet. The macro code is written in Visual Basic. It starts JMP in a visible state when the Microsoft Excel worksheet is initially opened. The Microsoft Excel worksheet is then imported into JMP using the ODBC automation interface. Once the worksheet data is in JMP, changes to individual worksheet cells are sent to JMP and changed in the JMP data table.

The first time a row value in Microsoft Excel changes, JMP generates a Control Chart. Subsequent changes to the Microsoft Excel worksheet result in changes to the Control Charts. This is because Control Chart output is dynamically linked to the JMP data table, which in this example is dynamically updated by Microsoft Excel. Every fifth time the Microsoft Excel worksheet changes, a method is called in JMP to generate a .PNG file for the Control Chart. This allows users without JMP to view the output through a web browser. Finally, when the Excel worksheet closes, JMP shuts down through automation.

Begin by opening Microsoft Excel. To create a Visual Basic script for a Microsoft Excel workbook, select **Visual Basic** from the Developer ribbon. The Visual Basic editor opens in a separate

window. On the left side of the Visual Basic editor, there is a pane entitled **VBA Project.** This pane shows the sheets that might have Visual Basic code associated with them, as well as the workbook itself.



Code written for the workbook usually works for any of the sheets within the workbook.

There are three sections involved in the coding for this example. First, there are some variables that are global in scope that are declared in the `module1.bas` file. This allows these variables to be referenced in other code modules. A module can be inserted into the Visual Basic project by context-clicking on the VBA project icon and selecting **Insert > Module**. Type the following code into the module. The code declares instances of a JMP application, a JMP data table, and a flag to keep track of whether a document is open or not.

```
Public MyJMP as JMP.Application     'The JMP Application Object
Public DT As JMP.DataTable          'The JMP Data Table object
Public DocOpen as Boolean           'A flag indicating "JMP Table Open"
```

The next segment updates JMP when cells in the Microsoft Excel worksheet change. It is called automatically because Microsoft Excel generates the `Worksheet_change` event whenever a cell is changed, deleted, or added.

The Microsoft Excel VBA Project Browser shows the sheets that are currently part of the workbook. The code below should be placed in the sheet that sends data to JMP. Double-click on the sheet icon in the VBA Project Window to bring up the code for that particular sheet.

```
Private Sub Worksheet_change(ByVal Target as Range)
      Dim Col as JMP.Column
      If(DocOpen) Then
            If(Target.Row = 1) Then
                  Return
            End If
            If(DT.NumberRows < Target.Row - 1) Then
                  DT.AddRows Target.Row - DT.NumberRows - 1, Target.Row
            End If
            If(Not IsArray(Target.Value) And Not IsEmpty(Target.Value))
Then
                  Set Col = DT.GetColumnByIndex(Target.Column)
                  Col.SetCellVal Target.Row - 1, Target.Value
            End If
      End If
End Sub
```

This code first checks to make sure JMP has a data table open. If the change is happening to the first row, then it is ignored because this is the column name in JMP. So, if a column name is

changed in Microsoft Excel, the corresponding change is *not* reflected in JMP. Code that would deal with heading changes could be inserted here, but is omitted in this example.

Next, if the row that has changed is beyond the number of rows that JMP is currently tracking in the data table, then the `AddRows` method is called to create more rows.

Finally, if the operation is on a single value and does not appear to signal a deletion, the JMP data table cell value is changed to the value that is passed into `Worksheet_Change.`

The main module is associated with the workbook. In the VBA Project Browser, the workbook code area is typically assigned the name `ThisWorkbook`, but this name can be easily changed. The following code goes into this area.

```
‘Public(Global Variables) that all Workbook subroutines can access
Public Counter As Integer ‘counter to update Control Chart every 5
changes
Public JMPDoc As JMP.Document ‘instance of JMP Document
Public CChart As JMP.ControlChart ‘instance of Control Chart
Public ChartOpen as Boolean ‘Flag to set if chart is open
Public DB As AUTODB

‘Shut Down JMP before closing the workbook
Private Sub Workbook_BeforeClose(Cancel as Boolean)
      DocOpen = False
      MyJMP.Quit
End Sub

‘As soon as the workbook is opened via File Open, load JMP for
Automation
Private Sub Workbook_Open( )
      Set MyJMP = CreateObject(“JMP.Application”) ‘Create an instance of
JMP
      MyJMP.Visible=True ‘Make this instance of JMP visible
      Counter = 0 ‘initialize counter that counts changes
      DocOpen = False ‘no document open yet
      ChartOpen = False ‘no charts open yet, either

      ‘CHANGE THIS PATH TO POINT TO THE EXCEL WORKSHEET
      Set DB = MyJMP.NewDatabaseObject
      DB.Connect ("DSN=Excel Files;DBQ=C:\Book2.xls;")
      Set DT = DB.ExecuteSQLSelect("SELECT * FROM ""Sheet1$""")
      DB.Disconnect
      Set JMPDoc = DT.Document
      DocOpen = True ‘Set flag to say that the document is open
End Sub

‘This is the most important part.
```

```
'After the first piece of data has been changed, generate a control
chart.
'After every 5 changes to Excel worksheet cells, generate a new PNG of
the Control Chart.

Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Source As
Range)
      Counter = Counter + 1
      'Save the control chart to a PNG every time 5 elements get updated
      If (Counter Mod 5 = 0 Or Counter = 1) Then

            'If the Control Chart has not been created yet, do so
            If Not (ChartOpen) Then
                  Set CChart = JMPDoc.CreateControlChart 'create chart
                  CChart.LaunchAddProcess "Column 1" 'Add column
                  CChart.LaunchAddSampleUnitSize 5
                  CChart.LaunchSetChartType jmpControlChartVar
                  CChart.Launch 'launch the chart
                  ChartOpen = True 'set flag to remember that a chart is
open
            End If
            CChart.SaveGraphicOutputAs "C:\ControlChart.png", jmpPNG
      End If
End Sub
```

The `Workbook_Open` subroutine is called when the Microsoft Excel table is initially loaded. It initializes some variables, starts JMP, and tells JMP to open (through ODBC) the same Microsoft Excel file that is currently loaded into Microsoft Excel 2013. Note that JMP opens the Microsoft Excel file as a database object rather than opening it as a file. This is necessary because JMP does not open a file that is already open in another application.

The `Workbook_Change` event is generated every time a user changes the data in any cell in any worksheet in the workbook. This sample assumes that there is only one active worksheet in the workbook. The first time the user changes a cell value in the worksheet, the `Workbook_Change` subroutine creates a Control Chart in JMP using the current data table.

In this sample, the `Workbook_Change` subroutine also creates a PNG graphic file of the Control Chart output and updates it on the disk every fifth time a change is made to the workbook. This just gives some ideas on how Microsoft Excel events and JMP automation can be used together to create output.

Finally, the `Workbook_BeforeClose` subroutine is invoked when the Microsoft Excel workbook is closed, but before the window goes away. The code within this subroutine instructs JMP to close down as well.

Note that there are some limitations in this method. This example is good if the only activities that occur with the data are additions or changes. The Microsoft Excel `Worksheet_Change` event is very limited in the reporting that it provides. In particular, cell-by-cell updating of a JMP data

table can be difficult in instances where deletion, drag and drop, or block replication needs support.

If these are problem cases, it is probably better to rely on a brute-force approach. One way is to reload the data into JMP every time a certain number of changes occur. An example is shown here.

```
Private Sub Workbook_SheetChange(ByVal Sh as Object, ByVal Source as
Range)
      Counter = Counter + 1
      If (Counter Mod 10 = 0) Then
            ‘If there is a previous chart of Table opened, close it
first
            If(DocOpen) Then
                  JMPDoc.Close False, ""
                  CChart.CloseWindow
            End If

            Set JMPDoc = MyJMP.OpenDocument(InstallDir + "C:\BOOK1.XLS")
            Set DT = JMPDoc.GetDataTable
            DocOpen = True

            ‘Now, create the control chart.
            ‘This one is keyed to the data in "Column 1".
            ‘If 5 or more values are changed,
            ‘JMP should generate a new Chart and save it as a
            ‘PNG file to disk.
            ‘The PNG file can be viewed with Internet Explorer.

            Set CChart = JMPDoc.CreateControlChart
            CChart.LaunchAddProcess "Column 1"
            CChart.LaunchAddSampleUnitSize 5
            CChart.LaunchSetChartType jmpControlChartVar
            CChart.Launch
            CChart.SaveGraphicOutputAs "C:\ControlChart.png", jmpPNG
      End If
End Sub
```

This sample reloads the data every time there are 10 changes to the Microsoft Excel Workbook. First, it removes JMP Control Charts and data tables that were previously created. Next, it loads the new data and creates a Control Chart.

This sample works best for small amounts of data. If very large Microsoft Excel files are involved, this approach is not efficient because of the reloading of the table into JMP.

Automating JMP through Visual C++

Using C or C++ to create an automation client can be a long, tedious task. However, if you use the support provided by MFC in Microsoft Visual C++, the task is considerably easier. There are several steps that must be performed in order to get to a state where you can launch the automation server application (JMP in this case). The Microsoft sample application CALCDRIV also shows a MFC-based automation client. CALCDRIV is typically included with Visual C++, and on MSDN CDs.

`AutoClient` shows how to start up JMP and drive a Bivariate analysis and the data table. The sample is much smaller than any of the Visual Basic samples, but the mechanics behind all the automation calls that you might want to use are the same as the examples with Bivariate and the data table. The following steps are based on the Visual C++ Version 5.0 UI.

Steps for Automating JMP

1. Create your application, either manually or through App Wizard. Specify support for OLE automation. Even if you are not automating your own application, you need to include the OLE headers and initialization code. If you are retrofitting an existing application, you need to make sure that you include OLE support. This usually means including afxole.h in your application, and calling `AfxOleInit()` in your application InitInstance routine. Consult the MFC OLE documentation for details about this.
2. Bring up the Class Wizard and select the Automation tab. Select the **Add Class** drop-down list and then the **From a Type Library** option. Navigate to the JMP install directory until you find `JMP.TLB`. Select this type library.
3. You are prompted to confirm the classes that you want to use in your project. If you are unsure what objects (and interfaces) that you want, select them all by pressing Shift and clicking. Select the names for the files where the class wizard generates interface stubs and header information. Class Wizard is generating wrapper classes based on the MFC ColeDispatchDriver class. This gives you easy access to the OLE Invoke automation function without having to know a lot of the technical details. Select **OK**. Class Wizard generates the two files (.h and .cpp). You should include the .h file in whatever .cpp files use the JMP automation objects. For example, your View class implementation file.
4. The Class View of your Workspace now shows the Interface classes that you have imported. You can examine the methods and properties for each class through this class view.
5. To start JMP, define a variable of type *`IJMPAutoApp`* that persist for the length of the automation session. Call `CreateDispatch` on this variable, passing in the JMP ProgID ("JMP.Application") as the lone parameter. At this point, when the code executes JMP starts.
6. Call `SetVisible(TRUE)` on the JMP object created in step 5. If you do not want to see JMP execute, do not do this step. However, for debugging it is necessary.
7. Now you can use the JMP application object to spawn further objects, which themselves can spawn more objects. The first thing you probably want to do is load a Data table. To load an existing JMP data table, call the `OpenDocument` method on the JMP object created in step 5. If successful, this method returns a dispatch pointer that can be attached to an object of type IJMPDoc using the `AttachDispatch` method.

8. The IJMPDoc object provides the methods to launch the analysis and graphing platforms. Once you create an analysis and attach the dispatch pointer, you can specify the data table columns to use in the analysis and then you can launch it. Once the analysis is launched, you can manipulate it using the properties and methods specific to that particular type of analysis. Code that is taken from the sample application that describes steps 5–8 is shown below:

Example Program

```
//Note, no error handling is done in this example
IJMPAutoApp m_DispDriver;
IJMPDoc     m_Doc;
IAutoBivar  m_Bivar;
IAutoFit    m_FitLine;

//Create the initial dispatch driver that uses the IJMPAutoApp
//interface specification (taken from jmpauto.h)
m_DispDriver.CreateDispatch("JMP.Application");

if (m_DispDriver)
{
      //If JMP successfully started, make it visible
      m_DispDriver.SetVisible(TRUE);

      //Now open a data table as a document. The document interface
      //pointer that is returned is then attached to our Doc dispatch
      //driver class that uses the IJMPDoc interface specification.
      m_Doc.AttachDispatch(m_DispDriver.OpenDocument(
            "C:\\JMPDATA\\BIGCLASS.JMP"));
}
      //First, call CreateBivariate on the Doc interface to create
      //a dispatch object to a Bivariate analysis. If there is already
      //a previous dispatch interface in m_Bivar, MFC releases it
      //in AttachDispatch.
      m_Bivar.AttachDispatch(m_Doc.CreateBivariate());

      //Now add Height and Weight as the columns to analyze
      m_Bivar.LaunchAddX("Height");
      m_Bivar.LaunchAddY("Weight");

      //Launch the analysis
      m_Bivar.Launch();

      //Create a FitLine. Since the Fit can be automated, attach the
      // dispatch pointer that is returned from FitLine()
```

```
// to a DispatchDriver object
m_FitLine.AttachDispatch(m_Bivar.FitLine());

//Now do a few more fits. This example does not automate these fit
//objects, although they do support automation.
m_Bivar.FitPolynomial(3.0);
m_Bivar.FitSpline(1000.0);

//Now manipulate the first FitLine object
m_FitLine.ConfidenceFit(TRUE);
m_FitLine.ConfidenceIndividual(TRUE);
```

Automating JMP in Visual C#

This section assumes you are using Visual Studio 2008 or later.

Starting a JMP Application

The first step to automating JMP in C# is to start JMP. Before you can do that, you need to add the JMP library to your resources. JMP provides a library that allows the user to use automation controllers to display the properties and methods that JMP uses.

To add the JMP library, follow these steps:

1. Select **Project > Add Reference** and then click the **COM** tab.
2. Scroll down, select **JMP**, and then click **OK**.
3. To verify that JMP has been added, select **View > Object Browser**.

   You should see `Interop.JMP`.
4. Now that you can access the type library information, write the necessary code to open JMP. In global declarations for the project, create a variable of type *`JMP.Application`*.

   ```
   private JMP.Application myJMP;
   ```
5. Create a JMP session and make it visible.

   ```
   myJMP = new JMP.Application();
   myJMP.Visible = true;
   ```

Launching an Analysis

This example shows how to create a Bivariate analysis of the Big Class.jmp sample data.

To launch the analysis, follow these steps:

1. Create a variable for the document.

   ```
   JMP.Document doc;
   ```
2. Open the `Big Class.jmp` sample data.

   ```
   doc = myJMP.OpenDocument("c:\Program Files\SAS\JMP\17\Samples\Data\Big
   Class.jmp");
   ```

3. Define a variable for the Bivariate analysis.

```
JMP.Bivariate biv;
```

4. Create the Bivariate object,.

```
biv = doc.CreateBivariate();
```

5. Add the necessary columns and values to specify which data to use. For Bivariate, you need to launch the platform and specify *x* and *y* values.

```
biv.LaunchAddX("weight");
biv.LaunchAddY("height");
```

6. Launch the Bivariate platform.

```
biv.Launch()
```

7. Create a Line of Fit to manipulate the data.

```
JMP.Fit fLine;
fLine = biv.FitLine();
```

The `FitLine` method creates an object of type `Fit`. This object has methods and properties of its own, which can be manipulated.

8. Show Line of Fit options such as Confidence Fit, Line of Fit, and Plot Residuals using the `fline.`*`function`*`(`*`Boolean`*`)` function:

```
fLine.ConfidenceFit(true);
fLine.LineOfFit(true);
fLine.PlotResiduals(true);
```

In this manner, you can manipulate many of the platforms, data table, and data access capabilities within JMP. The "Application Object Reference for Automating JMP on Windows" on page 49 describes all of the methods and properties that JMP exposes.

Starting a Specific JMP Version

Windows automation does not allow you to choose the specific version of the application that it starts up. The behavior is as follows:

- If JMP is not open, the automation program opens the version of JMP to which your file associations are set.
- If you haven't restored all of the Windows registry settings to factory defaults for a particular JMP installation, the version of JMP that was last installed is used.
- If an instance of JMP is already open, the automation program uses the running instance regardless of the version of JMP.

Application Object Reference for Automating JMP on Windows

The following sections contain details for the methods and properties that JMP exposes on Windows to automation clients such as Visual Basic, Visual C++, and Visual C#.

Constants

Each constant represents items used with specified commands.

Bivariate Platform Constants

bivarFitTransformConstants

Used in

```
Bivariate.FitTransformed()
```

Values

```
None
Log
Sqrt
Square
Reciprocal
Exp
```

bivarOrthogonalFitConstants

Used in

```
Bivariate.FitOrthogonal()
```

Values

```
Estimated Variances
Equal Variances
Fit Y to X
Specified Variance Ratio
```

fitLoessLambdaConstants

Used in

```
Bivariate.FitLoessWithParms()
```

Values

```
Linear
Quadratic
```

Chart Platform Constants

The original Chart platform in JMP has been deprecated. Automation code that calls the Chart platform will now use the equivalent Graph Builder commands to generate output. There may be differences in appearance from prior versions.

chartChartTypeConstants

Used in

```
chart.SpecifyType()
```

Values

```
Bar
Line
Needle
Point
Pie
```

chartOrientConstants

Used in

```
Chart.Orientation()
```

Values

```
Horizontal
Vertical
```

chartStatConstants

Used in

```
Chart.LaunchAddY()
```

Values

```
Data
N
% of Total
N Missing
Min
Max
SumWgt
Sum
Mean
Standard Deviation
Standard Error
Median
Range
Quantiles
Variance
CV
```

Cluster Platform Constants

clusterColormapConstants

Used in

```
Cluster.ColorMap()
```

Values

```
No Map
Green to Black to Red
Green to White to Red
White to Black
Blue to Gray to Red
Blue to Green to Red
Spectral
```

clusterDistanceConstants

Used in

```
Cluster.LaunchSpecifyDistanceFormula()
```

Values

```
Average
Centroid
Ward
Single
Complete
```

clusterOrientationConstants

Used in

```
Cluster.SetOrientation()
```

Values

```
Left
Right
Top
Bottom
```

Column Constants

colDataSourceConstants

Used in

```
Column.GetDataSource()
```

Values

```
Data
Formula
```

colDataTypeConstants

Used in

```
TextImport.SetColumnType()
```

```
Column.DataType
DataTable.NewColumn()
```

Values

```
Unknown
Numeric
Character
RowState
```

colFormatConstants

Used in

```
AxisBoxFormat
AxisBoxScale
Column.OutputFormat
Column.InputFormat
```

Values

```
Best
Short
Long
Abbrev
Date/Hr/Min
Date/Hr/Min/Sec
Days/Hrs/Mins
Days/Hrs/Mins/Secs
MMDDYYYY
MM/YYYY
DD/MM/YYYY
DDMMYYYY
DDMonYYYY
DD/MM/YYYY HH:MM
DD/MM/YYYY HH:MM:SS
YYYY/MM/DD
YYYYMMDD
H:M:S
MonDDYYYY
MonDDYYYY H:M
MonDDYYYY H:M:S
DDMonYYYY H:M
DDMonYYYY H:M:S
YYYY/MM/DD H:M
YYYY/MM/DD H:M:S
MM/DD/YYYY H:M
MM/DD/YYYY H:M:S
```

colModelTypeConstants

Used in

```
Datatable.NewColumn()
Column.ModelType
```

Values

```
Continuous
Ordinal
Nominal
```

colReorderConstants

Used in

```
DataTable.ReorderColumns()
```

Values

```
Original
By Name
By Datatype
By Modeling Type
Reverse
```

colRoleConstants

Used in

```
Column.SetRole()
```

Values

```
None
X
Y
Weight
Freq
```

colValidationConstants

Used in

```
Column.GetValiation()
```

Values

```
Unknown
None
List
Range
```

Control Chart Platform Constants

jmpControlChartAlarms

Used in

```
ControlChart.SetAlarm()
```

Values

```
Write
Speak
Write with Explanation
Speak with Explanation
```

jmpControlChartConstants

Used in

```
ControlChart.LaunchSetChartType()
```

Values

```
Variable
IR
P
N
C
U
UWMA
EWMA
Cusum
LeveyJennings
Presummarize
```

jmpControlChartRules

Used in

```
ControlChart.WestgardRule()
```

Values

```
All Rules
Rule 1 2S
Rule 1 3S
Rule 2 2S
Rule R 4S
Rule 4 1S
Rule 10 X
```

Data Table Constants

dtJoinConstants

Used in

```
DataTable.Join()
```

Values

```
By Row Number
Cartesian
Matching Columns
```

dtSummaryStatConstants

Used in

```
DataTable.AddToSummaryStatList()
```

Values

```
Data
N
% Of Total
N Missing
Min
Max
Sum Wgt
Sum
Mean
Variance
```

```
Std Dev
Std Err
Median
Range
Quantiles
CV
```

summaryStatColNameConstants

Used in

```
DataTable.SummarySetStatColumnFormat()
```

Values

```
StatColumn Format
Column Format
Stat of Column Format
Column Stat Format
```

Discriminant Constants

discrimCanonicalOptions

Used in

```
Discriminant.CanonicalOptions()
```

Values

```
Show points
Show ellipses
Show rays
Show contours
Show details
Save canonical scores
Color points
```

discrimScoreOptions

Used in

```
Discriminant.ScoreOptions()
```

Values

```
Show interesting rows
Show all distances
Show all probabilities
Show classification counts
Select misclassified rows
Save formulas
```

discrimPriorsOptions

Used in

```
Discriminant.SpecifyPoints()
```

Values

```
Equal Probabilities
Proportional to Occurrence
```

Distribution Platform Constants

distributionFitQuantilePlotConstants

These constants represent items that can be added to a Quantile Plot after a requested fit.

Used in

```
DistribFit.QuantilePlotAction()
```

Values

```
Rotate
Confidence Limits
Line Of Fit
Mean Reference Line
Probability Labels
```

distributionSaveConstants

These constants represent items that can be saved from the Distribution platform.

Used in

```
Distribution.Save()
```

Values

```
Level Numbers
Level Midpoints
Ranks
Ranks Averaged
Prob Scores
Normal Quantiles
Standardized
Spec Limits
```

fitDistribConstants

Used in

```
Distribution.FitDistribution()
```

Values

```
Normal
Log Normal
Weibull
Weibull With Threshold
Extreme Value
Exponential
Gamma
Beta
Poisson
SmoothCurve
GammaPoisson
GLog
```

DOE Constants

doeChangeDifficultyConstants

Used in

```
doe.SpecifyChangeDifficulty
```

Values

```
Easy
Hard
```

doeFactorTypes

Used in

```
DOECustom.AddFactor()
```

Values

```
Continuous
Categorical
Mixture
```

doeModelTypes

Used in

```
DOE.MakeModel()
```

Values

```
Linear
Interactions
RSM
```

doeOptimalityConstants

Used in

```
DOE.OptimalityCriterion()
```

Values

```
Recommended
D-Optimal
I-Optimal
```

doeResponseTypes

Used in

```
DOE.AddResponse()
```

Values

```
Maximize
Match Target
Minimize
None
```

Fit Model Platform Constants

fitModelDistributionConstants

Used in

```
FitModel.LaunchSpecifyEmphasis()
```

Values

```
Weibull
LogNormal
Exponential
```

fitModelEffectAttributeConstants

Used in

```
FitModel.LaunchSpecifyAttributesForSelectedEffects()
```

Values

```
Random Effect
Response Surface Effect
LogVariance Effect
Mixture Effect
Excluded Effect
```

fitModelEmphasisConstants

Used in

```
FitModel.LaunchSpecifyEmphasis()
```

Values

```
Effect Leverage
Effect Screening
Minimal Report
```

fitModelMacroEffectConstants

Used in

```
FitModel.LaunchAddMacroEffect()
```

Values

```
Full Factorial
Factorial to Degree
Factorial Sorted
Response Surface
Mixture Response Surface
Polynomial to Degree
Scheffe Cubic
```

fitModelPersonalityConstants

Used in

```
FitModel.LaunchSpecifyPersonality()
```

Values

```
Standard Least Squares
Stepwise
Manova
Loglinear Variance
Nominal Logistic
Ordinal Logistic
Proportional Hazard
Parametric Survival
```

fitModelRandomEffectMethods

Used in

```
FitModel.LaunchSpecifyRandomEffectMethod()
```

Values

```
REML - Recommended
EMS  - Traditional
```

fitModelRowDiagConstants

Used in

```
FitModel.LaunchSpecifyAttributesForSelectedEffects()
```

Values

```
Plot Actual by Predicted
Plot Effect Leverage
Plot Residual by Predicted
Plot Residual by Row
Press
DurbinWatson
```

fitModelSaveColumnConstants

Used in

```
FitResponse.SaveColumns()
```

Values

```
Prediction Formula
Predicted Values
Residuals
Mean Confidence Interval
Individual Confidence Interval
Studentized Residuals
Hats
Standard Error of Predicted
Standard Error of Residual
Standard Error of Individual
Effect Leverage Pairs
Cook's D Influence
Standard Error of Predicted Formula
```

fitModelTransforms

Used in

```
FitModel.LaunchAddXEffectWithTransform()
FitModel.LaunchAddYWithTransform()
```

Values

```
No Transform
Log
Square Root
Square
Recip
Exponential
Arrhenius
Arrhenius Inverse
```

fitStepDirectionConstants

Used in

```
FitStepwise.SetDirection()
```

Values

```
Forward
Backward
Mixed
```

fitStepRulesConstants

Used in

```
FitStepwise.SetRules()
```

Values

```
Combine
Restrict
No Rules
Whole Effect
```

Item Analysis Constants

itemAnalysisModelConstants

Used in

```
ItemAnalysis.LaunchSpecifyModel()
```

Values

```
Logistic 1PL
Logistic 2PL
Logistic 3PL
```

JMP Constants

These constants are available for application-level commands or for all platform commands.

axisBooleanConstants

Used in

AnalysisPlatform.AxisBoxBooleanOption()

Values

```
Show Major Ticks
Show Minor Ticks
Show Major Grid Lines
Show Minor Grid Lines
Show Labels
Rotate Labels
```

axisIntervalConstants

Used in

AnalysisPlatform.AxisBoxInterval

Values

```
Numeric
Year
Month
Week
Day
Hour
Minute
Second
```

axisLineRefConstants

Used in

AnalysisPlatform.AxisBoxAddRefLine()

Values

```
Solid
Dashed
Dotted
```

axisNumericOptionConstants

Used in

AnalysisPlatform.AxisBoxNumericOption()

Values

```
Axis Minimum
Axis Maximum
Number of Minor Ticks
Increment between Ticks
```

axisScaleConstants

Used in

AnalysisPlatform.AxisBoxScale()

Values

Linear
Log

commFlowControlConstants

Used in

DataFeed.SetCommParms()

Values

None
DTR/DSR
RTS/CTS
XOn/XOff

commParityConstants

Used in

DataFeed.SetCommParms()

Values

None
Even
Odd

frameMarkerSizes

Used in

AnalysisPlatform.FrameBoxSetMarkerSize()

Values

Dot
Small
Medium
Large
XL
XXL
XXXL

internetItemTypes

Used in

Application.InternetOpenItem()

Values

HTML
Edit HTML/Text
HTML with tags stripped
JMP Table

```
JMP Table from HTML
Run JSL file on Web
```

jmpColorConstants

Used in

AnalysisPlatform.AxisBoxAddRefLine()
AnalysisPlatform.FrameBoxSetBackColor()
Chart.OverlayColor()
ControlCharts.ConnectColor()
ControlCharts.CenterColor()
ControlCharts.LimitsColor()
Surface.SetItemColor()

Values

```
Black
Red
Green
Blue
Orange
Purple
Yellow
Magenta
```

jmpGraphicsFormats

Used in

Journal.SaveAsHTML()
Journal.SaveAsRTF()
AnalysisPlatform.SaveGraphicOutputAs()
AnalysisPlatform.SaveGraphicItem()

Values

```
PNG Format
JPEG Format
Windows Metafile
```

jmpMarkerConstants

Used in

Overlay.YOverlayMarker()

Values

```
Dot
Plus
X
Hollow Square
Diamond
Triangle
Y
Z
Hollow Circle
Hollow Flat Rectangle
Hollow Tall Rectangle
Star
Solid Circle
```

```
Solid Flat Rectangle
Solid Tall Rectangle
Solid Square
```

jmpScriptConstants

Used in

*AnalysisPlatform.*ScriptAction()

Notes:Save To File is no longer supported; value 2 is meaningless.

Values

```
Redo Analysis
Save To File
Save To Data Table
Save To Report
Save To Window
```

jmpWindowTypeConstants

Used in

Application.CloseWindowsOfType()

Values

```
Datatables
Reports
Journals
JSL Output
Scripts
```

nomAxisActions

Used in

nomAxisBooleanOption()

Values

```
Rotate Ticks
Dividers
Lower Frame
```

printOrientConstants

Used in

*AnalysisPlatform.*SetPrintOrientation()

Values

```
Portrait
Landscape
```

Neural Constants

neuralControlConstants

Used in

```
Neural.ControlPanelOptions()
```

Values

```
Log the Tours
Log the Iterations
Log the estimates
Save the iterations
```

Oneway Platform Constants

OnewayCompareConstants

These constants represent the four multiple comparison methods for a oneway analysis

Used in

```
Oneway.CompareMeans()
```

Values

```
Each Pair
All Pairs
With Best
With Control, Dunnetts
```

OnewayDisplayConstants

These constants are options that can be toggled on and off in a oneway report.

Used in

```
OneWay.DisplayOptions()
```

Values

```
All Graphs
Points
Quantile Boxes
Means Diamonds
Means Dots, Error Bars
Grand Mean
Standard Deviation Lines
Comparison Circles
Connect Means
X Axis Proportional
Jitter
Matching Lines
Quantile Fit Lines
V Axis
H Axis
Mean Lines
Mean CI Lines
Mean of Means
Points Spread
```

OnewayNonParConstants

These constants represent the three nonparametric tests in a oneway analysis.

Used in

```
Oneway.Nonparametric()
```

Values

```
Wilcoxon
Median
van der Waerden
```

OnewaySaveConstants

These are the three options for saving values from a oneway report. Template and Normal Quantiles are the same option: `Template` is the old term, and `Normal Quantiles` matches the new term used in the platform.

Used in

```
Oneway.Save()
```

Values

```
Centered
Standardized
Template
Normal Quantiles
```

Overlay Constants

The original Overlay platform in JMP has been deprecated. Automation code that calls the Overlay platform will now use the equivalent Graph Builder commands to generate output. There may be differences in appearance from prior versions.

overlayLineStyleConstants

Used in

```
Overlay.LineOptions()
```

Values

```
Solid
Dotted
Dashed
Dash Dot
Dash Dot Dot
```

overlayLineThicknessConstants

Used in

```
Overlay.LineOptions()
```

Values

```
Regular
Thicker
Thickest
```

Partition Constants

partitionCriterionConstants

Used in

```
partition.Criterion()
```

Values

```
Maximize Split Statistic
Maximize Significance
```

partitionDisplayConstants

Used in

```
partition.DisplayOptions()
```

Values

```
Show Points
Show Tree
Show Graph
Show Split Stats
Show Split Candidates
Sort Split Candidates
Show Split Bar
Show Split Probability
```

partitionMissingConstants

Used in

```
partition.MissingValueRule()
```

Values

```
Closest
Random
```

partitionSaveColumnsConstants

Used in

```
partition.SaveColumns()
```

Values

```
Save Residuals
Save Predicteds
Save Leaf Numbers
Save Leaf Labels
Save Predicted Formula
Save Leaf Number Formula
Save Leaf Label Formula
```

Row Constants

rowStateConstants

Used in

```
DataTable.GetNumberOfRowsByRowState()
```

Values

```
Selected
Hidden
Excluded
Labeled
```

rowSelectWhereHow

Used in

```
Datatable.SelectRowsWhere()
```

Values

```
Clear Previous Selection
Extend Current Selection
Select From Within Current Selection
```

rowSelectWhereOperations

Used in

```
Datatable.SelectRowsWhere()
```

Values

```
Equals
Not Equals
Greater Than
Greater Than or Equals
Less Than
Less Than or Equals
Contains
Does Not Contain
```

Scatterplot Matrix Constants

scatterMatrixFormatConstants

Used in

```
ScatterplotMatrixPlatform.LaunchSpecifyMatrixFormat()
```

Values

```
Lower Triangular
Upper Triangular
Square
```

Surface Constants

surfaceColorConstants

Used in

```
Surface.SetItemColor()
```

Values

```
Grid Color
Mesh Color
Axis Color
Value Color
Name Color
Contour Color
```

surfaceDisplayConstants

Used in

```
Surface.DisplayOptions()
```

Values

```
Show X Axis
Show Y Axis
Show Z Axis
Show X Value
Show Y Value
Show Z Value
Show X Name
Show Y Name
Show Z Name
Show X Grid
Show Y Grid
Show Z Grid
Show Lights Border
Show Control Panel
Show Surface
Show Mesh
Show Contour
Lock Z Scale
Show Data Points
```

Survival Constants

competingCauseConstants

Used in

```
Survival.CompetingCauseAction()
```

Values

```
Omit Causes
Save Cause Coordinates
Weibull Lines
Hazard Plot
```

Text Import Constants

jmpTIEndOfFieldConstants

Used in

```
TextImport.SetEndOfFieldOptions()
```

Values

```
Tab
Space
Spaces
Comma
```

jmpTIEndOfLineConstants

Used in

```
TextImport.SetEndOfLineOptions()
```

Values

```
Carriage Return+Line Feed
Carriage Return
Line Feed
Semicolon
```

Time Series Platform Constants

timeSeriesConstraintConstants

Used in

```
TimeSeries.SmoothingModel()
```

Values

```
ZeroToOne
Unconstrained
Stable
```

timeSeriesModelConstants

Used in

```
TimeSeries.SmoothingModel()
```

Values

```
Simple Exponential
Double Exponential
Linear Exponential
Damped Trend
Seasonal Exponential
Winters Method
```

Variability Chart Platform Constants

varVarianceComponentConstants

Used in

```
Variability.VarianceComponents()
```

Values

```
Nested
Crossed
Crossed then Nested
Nested then Crossed
```

Application Object

The Application object provides high-level support for running JMP and loading data tables and other files. It is the essential object that must be created in order to have an automation session.

Properties

Application

Returns a dispatch pointer to the JMP object, which you should already have if you are accessing the property.

FullName

Returns the short name of the JMP application as a string, for example, “JMP”.

Name

As with `FullName`, `Name` returns the short name of the JMP application as a string.

Parent

Returns the object that is the next level up. Since the application object is top level, it just returns the application object.

Visible

Sets the JMP session visible if set to `True` (1), invisible if `False` (0). The default is `False`.

Setting Visible to `True` affects all windows, both those created before and after changing this setting.

Methods

ClearLog()

Clears the contents of the Log window.

CloseAllWindows()

Closes all currently open windows.

CloseWindow()As Boolean

Closes the analysis window immediately, rather than waiting for JMP to Exit. Returns `True` if successful, `False` if not.

CloseWindowsOfType(`jmpWindowTypeConstants` windowType)

Closes all currently open windows of a given type, like `Journal` or `Datatable`. `jmpWindowTypeConstants` contains the available window types that may be closed.

CreateDOECustom () As DOECustom

Creates a `DOE Custom Design` object. This object is then invoked with methods to Add Factors, Add a Model, Create a Design and finally make a Table. Please read the section under automation of Design of Experiments for further information.

CreateTextImportObject(FileName As String, NumberColumns as Integer) As TextImport

Creates a `TextImport` object, which must then be set up with information on columns and rows. `FileName` is the full path of the file that will be imported, `NumberColumns` describes how many table columns are in the data. Returns a dispatch pointer to the new `TextImport` object.

EnableInteractiveMode(Flag as Boolean)

Lets you display information in message boxes during an automated process rather than in the log. This option effectively turns off the Batch mode processing.

GetLogContents() As String

Returns the current contents of the Log window as a String. The Log Window can be floating or docked. If the log is hidden, an empty string is returned.

GetJSLValue

Used to retrieve the value of a JSL global variable of type Integer, Double, String or a List whose elements are a heterogeneous mix of those three types of values. The return value is a VARIANT, which can contain the Integer, Double, String or an Array of Variant records.

The method declaration is:

```
GetJSLValue (VariableName As String) As Variant
```

A typical call in Visual Basic is:

```
result = GetJSLValue("MyJSLVariable")
```

Accessing results depends on the type returned, and might be something like this:

```
A = B * result;
```

Or if the value is a String:

```
MsgBox(result)
```

Or if the value is a List and you want the 3rd element:

```
MsgBox(result(2))
```

GetNumberOfAutomationDatatables()As Integer

Returns the number of currently open/viewable data tables within JMP.

GetRunCommandErrorString() As String

Allows the JSL error text to be retrieved after the existing `RunCommand(Command As String)` and `RunJSLFile(FileName As String)` methods have been run.

See also `HasRunCommandErrorString() As Boolean`.

An example of the Visual Basic code to access these methods is:

```
MyJMP.RunCommand (Text1.Text)
If (MyJMP.HasRunCommandErrorString) Then
      MsgBox (MyJMP.GetRunCommandErrorString)
End If
```

GetTableHandleFromIndex(Integer Index)

Returns the handle to the `DataTable` automation object given the index of a data table in the range from 1 to the number of data tables open within JMP.

GetTableHandleFromName(Name as String)As DataTable

Finds a data table based on its name, and returns a handle to the automation object of the table.

GetTableNameFromIndex(Integer Index)

Returns the name of the data table as a string given the index of a data table in the range from 1 to the number of data tables open within JMP.

HasRunCommandErrorString() As Boolean

Provides a simple way to query to see if there is error text at all, rather than checking for an empty string.

See also `GetRunCommandErrorString() As String`.

An example of the Visual Basic code to access these methods is:

```
MyJMP.RunCommand (Text1.Text)
If (MyJMP.HasRunCommandErrorString) Then
      MsgBox (MyJMP.GetRunCommandErrorString)
End If
```

HonorSessionSavePref(Flag as Boolean)

Under automation, session save is not performed on shutdown. Automation operations often recreate a certain state, and the session save confuses this. If session save is desired on shutdown during automation, call this method with a `True` parameter before calling the `Quit` method and

then JMP will follow the session save preference setting. Passing a parameter of `False` tells JMP to ignore the session save for automation.

InternetOpenItem(String URL, internetItemTypes openHow) As DataTable

Opens a text or binary file. The options include opening a HTML file in its raw form into a text editor, opening a HTML file as text with the HTML tags stripped out, opening a binary JMP file and opening a HTML file that contains TABLE tags (TABLE, TD, TR, etc.) as a JMP data table. The second parameter determines the action. For the last two methods, a pointer to a JMP data table automation object will be returned in the method invocation is successful, otherwise a NULL will be returned. For the first two methods, a NULL is always returned.

InternetOpenTextAsData(URL As String) As DataTable

Opens a Text file at the specified URL into a JMP script window, and then attempts to import the text in that Window as a JMP data table. The current preferences for Text Import are used for the text import phase. If successful, the function returns a pointer to a JMP data table that can be manipulated using the DataTable automation object methods. NULL is returned if the method fails.

An example of code for this is:

```
Set DT = MyJMP.InternetOpenTextAsData("www.sas.com/MyData/data.txt")
Dim Doc As JMP.Document
Set Doc = DT.Document
Doc.SaveAs ("c:\myData.jmp")
```

NewDatabaseObject() As AutoDB

Creates an object of type `AutoDB`, which is used for automating ODBC access to data.

NewDataFeed() As DataFeed

Creates a data feed object used to sample an external instrument hooked up to a serial port.

NewDataTable(FileName As String) As DataTable

Creates a new JMP data table, and returns the object so that it can be further automated.

OpenDocument(FileName As String) As Document

Opens a JMP data table as a Document.`OpenDocument(FileName As String) As Document` is a standard document access routine for automation applications, so it is provided in JMP. If a data table is loaded and the user wishes to manipulate the table contents, the Document method `GetDataTable` must be called to get a `DataTable` object. The `DataTable` object is what allows the contents to be changed.

Quit()

Shuts down JMP if no other automation applications are using it. Decrements the use count on JMP if other applications are automating it.

RunCommand(Command As String)

Runs JSL text that is provided in string form.

RunJSLFile(FileName As String)

Loads a JSL text file from disk given a valid path name, and then submits the text for execution within JMP.

SetCurrentDirectory (DirName As String) As Boolean

Sets the current directory within JMP. This allows the use of relative file names in other methods. This should be used with caution if existing automation client code assumes the use of the JMP installation directory.

ShowLogHonorPreferences()

This method is a variation of `ShowLog()`. The log is normally suppressed for Automation. However, if the JMP preferences indicate that the log should appear on startup, calling this function will show the log. The other log preferences are to show the log only when the user opens it or when JMP writes text to the log.

ShowStartupWindow()

Shows the JMP startup window (usually the Home window). If the Home Window is already showing, it is brought to the foreground.

ShowLog() As Boolean

Show the Log window. If the Log is already showing, nothing happens. Returns `True` if the Log is available, and `False` if it is not.

AUTODB Object

The `AUTODB` object provides a mechanism for accessing external data using ODBC. Some knowledge of SQL is necessary to do table manipulation.

Methods

Some knowledge of SQL is necessary to use these methods.

Connect(ConnectInformation As String)

Contains a connect string that will ultimately be used for an `SQLDriverConnect` call. An example is

```
DSN=oracledata;DBQ=data_o7555;UID=UserID; pwd=userPassword
```

See the automation example code for further examples.

Disconnect() As Boolean

Shuts down the connection.

ExecuteSql(SQLStatement As String) As Boolean

Executes the SQL statement and returns a boolean to indicate success or failure.

Notes:The SQL statement cannot be a Select statement that returns a record set. To send Select statements, use `ExecuteSqlSelect(SQLSelectStatement As String) As DataTable`**.**

ExecuteSqlSelect(SQLSelectStatement As String) As DataTable

Executes the SQL `Select` statement and returns a dispatch pointer to the newly loaded table.

Note: The SQL Statement must be a Select statement that returns a record set. For other SQL commands, use `ExecuteSql(SQLStatement As String) As Boolean`.

GetLastError()

Returns the error code from the last `Connect` or `ExecuteSQL` call.

OpenTable(TableName As String) As DataTable

Open the database table, and return a dispatch pointer to the JMP table that is created to hold the data.

SaveTable(TablePtr As DataTable, FileName As String)

Given a dispatch pointer to a JMP data table, save the table to the database using the name provided.

AxisBox

AxisBox commands enable you to manipulate axis settings.

Methods

AxisBoxAddLabel(Handle As Long, Label As String)As Boolean

Adds a label for the axis identified by `Handle`, returning `True` if successful, `False` if not.

AxisBoxAddRefLine(Handle As Long, Location As Double, Style As Short, Color As Short)As Boolean

Adds a reference line at the location specified by the `Location` numeric.The `Style` value can be obtained from the `axisLineRefConstants`, the color from `jmpColorConstants`.

AxisBoxBooleanOptions(Handle As Long, Option As Short, Flag As Bool)As Boolean

Provides a standard call to set the axis Boolean options. The options that can be specified are included in `axisBooleanConstants` and include `Show Major Ticks`, `Show Minor Ticks`, `Show Major Grid`, `Show Minor Grid`, `Show Labels` and `Rotated Labels`. For each option, a value of `True` for `Flag` turns the option on, `False` turns it off. A return value of `True` indicates success, `False` indicates failure.

AxisBoxFormat(Handle As Long, Format As Short)As Boolean

Specifies the format for the axis marks. The format values can be found in `colFormatConstants`. Examples include `Best` and `m#d#y`.

AxisBoxInterval(Handle As Long, Interval As Short)As Boolean

Specifies the units used for the Inc (Increment) value, e.g. `Numeric`, `Hour`, `Day`, and so forth.Values for the interval can be found in `axisIntervalConstants`.

AxisBoxNumberDecimals(Handle As Long, NumDecimals As Short)As Boolean

Specifies the number of decimals for the axis value format.

AxisBoxNumericOption(Handle As Long, Option As Short, Number As Double)As Boolean

Provides a standard call to set Axis numeric options. The options can be found in `axisNumericOptionConstants`, and include `Min`, `Max`, `Inc` (Increment between ticks), and `Minor Ticks`.

AxisBoxRemoveLabel(Handle As Long)As Boolean

Removes the axis label.

AxisBoxRevertAxis(Handle As Long)As Boolean

Attempts to revert the axis to its original settings. A return value of `True` indicates success, `False` failure.

AxisBoxScale(Handle As Long, Scale As Short)As Boolean

Changes the scale of the axis to either Log or Linear.The values for `Scale` can be obtained from `axisScaleConstants`.

Column Object

The Column object exposes an individual data table column to manipulation via automation. A column object pointer must first be obtained from the `DataTable` object with one of the `GetColumn` calls.

Properties

DataType

Property that indicates if the column data type is `Character`, `Numeric`, `RowState` or `unknown`. The value is part of the `colDataTypeConstants` definition.

FieldWidth

Property that indicates the width of the column field as an integer.

Notes:To specify a value for the `FieldWidth` **property, you should specify the value after you specify the** `OutputFormat` **property value. If you also specify the** `NumDecPlaces` **property, you should specify the** `FieldWidth` **after you specify the** `NumDecPlaces`**.**

InputFormat

Integer property that indicates the input format for the data. This is typically `Best` for numeric data, or one of the date/time formats for date/time data. The value is part of the `colFormatConstants` definition. The formats `Long` (Long date/time) and `Abbrev` (Abbreviated date/time) are invalid for input, and will generate an error.

OutputFormat

Integer property that indicates the output format for the data. This is typically `Best` for numeric data, or one of the date/time formats for date/time data. The value is part of the `colFormatConstants` definition.

Notes:If you set the `Column.OutputFormat` **property to** `JMP.colFormatConstants.colFormatCurrency`**, the number of decimal places is set to 2 (for US dollars). If at any time you set the** `Column.NumDecPlaces` **to a number, the** `OutputFormat` **property is overridden, and** `Column.OutputFormat` **is set to** `colFormatFixedDecimal`**. This means that there is no way to set currency to have a different number of decimal places through automation, since each overrides the other.**

Locked

A Boolean (True/False) property that indicates if the column is locked.

ModelType

Integer property that indicates whether the column model type is Continuous, Nominal or Ordinal. The value is part of the `colModelTypeConstants` definition.

Name

String property that gives the column name.

NumberRows

Integer property that indicates the current number of rows in the column.

NumDecPlaces

Integer property that indicates the number of decimal places that are allowed for a numeric column.

Note: To specify `NumDecPlaces` you must specify the `OutputFormat` is `colFormatFixedDec.` The `NumDecPlaces` property should be specified after the `OutputFormat` property. If you also specify

the FieldWidth property, you should specify NumDecPlaces between OutputFormat and FieldWidth.

Methods

AddFormula(JSLText As String)

Adds a formula to the column, given a valid formula specified using JSL.

AddValueLabelToList(Value as String, Label as String) As Boolean

Adds a value label to an automation list. The value label will not be assigned to the column until CommitValueLabels() As Boolean is called. In this way, several value labels can be accumulated for the column before they are assigned. Returns True for success, False for failure.

See also CommitValueLabels() As Boolean and RemoveValueLabels() As Boolean.

An example of using these methods in a Visual Basic application is:

```
' Using the data table "Big Class.jmp" already assigned in object DT

Dim ColSex As JMP.Column
Dim ColAge As JMP.Column

Set ColSex = DT.GetColumn("sex")
Set ColAge = DT.GetColumn("age")

ColSex.AddValueLabelToList "M", "Male"
ColSex.AddValueLabelToList "F", "Female"
ColSex.CommitValueLabels
ColAge.AddValueLabelToList "12", "Twelve"
ColAge.AddValueLabelToList "13", "Thirteen"
ColAge.AddValueLabelToList "14", "Fourteen"
ColAge.AddValueLabelToList "15", "Fifteen"
ColAge.AddValueLabelToList "16", "Sixteen"
ColAge.AddValueLabelToList "17", "Seventeen"
ColAge.CommitValueLabels
```

CommitValueLabels() As Boolean

Commits (adds) all the value labels to the columns that were previously assigned using AddValueLabelToList(Value as String, Label as String) As Boolean. After CommitValueLabels is called, the column will update with the new labels. Returns True for success, False for failure.

See also AddValueLabelToList(Value as String, Label as String) As Boolean and RemoveValueLabels() As Boolean.

An example of using these methods in a Visual Basic application is:

```
' Using the data table "Big Class.jmp" already assigned in object DT
```

```
Dim ColSex As JMP.Column
Dim ColAge As JMP.Column

Set ColSex = DT.GetColumn("sex")
Set ColAge = DT.GetColumn("age")

ColSex.AddValueLabelToList "M", "Male"
ColSex.AddValueLabelToList "F", "Female"
ColSex.CommitValueLabels
ColAge.AddValueLabelToList "12", "Twelve"
ColAge.AddValueLabelToList "13", "Thirteen"
ColAge.AddValueLabelToList "14", "Fourteen"
ColAge.AddValueLabelToList "15", "Fifteen"
ColAge.AddValueLabelToList "16", "Sixteen"
ColAge.AddValueLabelToList "17", "Seventeen"
ColAge.CommitValueLabels
```

Exclude() As Boolean

Excludes the column. This operation is a toggle, meaning that calling them once sets them and calling them a second time unsets them. A return value of `True` indicates success, `False` indicates failure.

GetCellVal(RowNumber As Integer) As String

Returns the value of a given row of the column as a string. If the value is really a numeric, any conversion must be done by the caller. Visual Basic can do this automatically if the receiving object is defined as a numeric.

GetDataSource() As Integer

Attempts to find out of the column is regular data, instrument data, or contains a formula. It returns an integer that is part of the `colDataSourceConstants` definition.

GetDataVector() As Variant

Retrieves a vector with all the elements of a column, passed as a Variant.

An example of using `GetDataVector` in Visual Basic is:

```
'Get All the names from the "Name" column.  Show the first 5 in a
'messagebox

Dim Col As JMP.Column
Set Col = DT.GetColumn("name")
nameArray = Col.GetDataVector
For i = 0 To 4
      MsgBox (nameArray(i))
```

```
Next i
```

GetFormula() As String

Retrieves the formula for the column in the form of a string.

GetRowStateVectorData

Returns the actual data that matches a rowstate criterion. This provides a one-call way to get the data, rather than using `GetRowStateVector` and then using the indices in a call to `SetCellVal(RowNumber As Integer, Value as String)`. This provides a potential performance gain as well. The method declarations look like:

```
GetRowStateVectorData(rowStateConstants state) As Variant
```

An example of using `GetRowStateVectorData` in Visual Basic:

```
Dim Col As JMP.Column
Set Col = DT.GetColumn("height")
myarray = Col.GetRowStateVectorData(rowStateSelected)'Get the vector of
selected row numbers
nElements = UBound(myarray)   'Get the upper bound of the returned array

For i = 0 To nElements        'Display the names from the selected rows
      MsgBox (myarray(i))
Next I
```

GetValidation() As Integer

Attempts to find out if the column has list, range, or no validation. Returns an integer that is part of the `colValidationConstants` definition.

InsertDataVector(Data As Variant, AfterRow As Long) As Boolean

Accepts an array of Doubles or Strings (based on the column type) and sets the cell values for the Column with those array values, starting after the row specified by the second parameter. This is an overwrite operation if data exists after the specified row.

For example, if the second parameter is a 5 and data exists in rows 6-10 and the data vector has 8 values, then rows 6-10 will be overwritten and 3 additional rows will be created for the remaining data.

If the second parameter is specified as 0, then the data will be *appended* to the beginning of the column. Additional rows will be created for the data. If you want to add data starting at row 1, then use `SetDataVector(Data As Variant) As Boolean` instead. If the second parameter is -1, then the data will be appended to the column after the last existing row. Rows will be created to accommodate the new data.

Hide() As Boolean

Hides the column. This operation is a toggle, meaning that calling them once sets them and calling them a second time unsets them. A return value of `True` indicates success, `False` indicates failure.

Label() As Boolean

Makes the column a label. This operation is a toggle, meaning that calling them once sets them and calling them a second time unsets them. A return value of `True` indicates success, `False` indicates failure.

RemoveValueLabels() As Boolean

Removes all the value labels associated with the column. This is done immediately, and the column should revert back to showing the original values as soon as this method is called. Returns `True` for success, `False` for failure.

See also `CommitValueLabels() As Boolean` and `AddValueLabelToList(Value as String, Label as String) As Boolean`.

ScrollLock() As Boolean

Scroll locks the column. This operation is a toggle, meaning that calling it once sets the lock and calling it a second time unsets the lock. A return value of `True` indicates success, `False` indicates failure.

SelectCellMissing(Index as Integer)

Sets the value of the cell at the specified row to a Missing Value.

SelectColumn(Flag as Boolean) As Boolean

Select (`Flag` is `True`) or de-select the column (`Flag` is `False`). A return value of `True` indicates success, `False` indicates failure.

SetCellVal(RowNumber As Integer, Value as String)

Sets the value of a row of the column, converting the value to numeric if the column contains numeric data.

SetDataVector(Data As Variant) As Boolean

Accepts an array of Doubles or Strings (based on the column type) and sets the cell values for the Column with those array values starting with row 1. If there are fewer rows than data values, then additional rows are created in the table. Returns a `True` for success, a `False` for failure.

SetCellMissing(Row As Integer)

Sets the cell that the row number specifies to a missing value.

SetCurrencyType(Type As ColCurrencyConstants)

Sets the type of currency for the entire column.

SetRole(RoleType As Integer) As Boolean

Sets the role type of the column using one of the values from `colRoleConstants`.

Common Analysis Functions

Each analysis and graphing platform shares a common set of functions. So, regardless of what analysis you are running you can call these methods.

Methods

You can use all of the Common Analysis methods with any analysis or graph platform.

CreateJournal() As Journal

Creates a journal and returns a pointer to the automation object the represents it. This object can then be used to save output to disk.

CopyGraphicItem(Handle As Long) As Boolean

Copies a metafile representation of the graphic sub-item identified by handle to the clipboard.

DisplayBoxAppend(SrcHandle as Long, AppendHandle as Long) As Long

Adds a display box to the end of another display box.

DisplayBoxPrepend(SrcHandle as Long, AppendHandle as Long) As Long

Adds a display box before another display box.

FrameBoxAddGraphicsScript(long handle, script As String) As Boolean

Associates a graphics script with a FrameBox, given its handle (returned from `GetGraphicItemByType`). This is just like the interactive dialog to add a graphics script to an analysis frame. The method returns `True` if it is successful, or `False` if there is a failure.

FrameBoxSetBackColor(long handle, jmpColorConstants color) As Boolean

Sets the background color within a FrameBox given its handle (returned from `GetGraphicItemByType`). The `jmpColorConstants` define the range of colors. The method returns `True` if it is successful, or `False` if there is a failure.

FrameBoxSetMarkerSize(long handle, `frameMarkerSizes` size) As Boolean

Sets the size of the markers within a Frame Box, given its handle (returned from `GetGraphicItemByType`). The `frameMarkerSizes` constants define the range of sizes. The method returns `True` if it is successful, or `False` if there is a failure.

FrameBoxTransparency(alpha as Double) As Boolean

Sets the transparency level of the points within the graph.

GetGraphicItemByName(ItemName as String) As Long

Returns a handle to a JMP display sub-item. An example of a sub-item would be the "Analysis of Variance" results table from a Bivariate analysis. This handle can be used be other operations described below.

GetGraphicItemByType(TypeName As String, ItemNumber As Integer) As Long

Returns a handle to a JMP display sub-item based on the item type. An example of a sub-item type would be the `PictureBox`. PictureBox number one might return a handle to an analysis plot. This handle can be used be other operations described below.

GetSubgraphicItemByName(Handle as Long, Name as String) As Long

Returns a handle to a display box.under the display box referenced by `Handle`. The display box returned is given by the title of the box. This is similar to the method `GetGraphicItemByName(ItemName as String) As Long`. The difference is the starting display box. `GetGraphicItemByName(ItemName as String) As Long` starts at the top of the display while `GetSubgraphicItemByName` is relative to the display box referenced by handle.

GetSubgraphicItemByType(Handle as Long, BoxName as String, BoxNumber as Short) As Long

Returns a handle to a display box under the display box referenced by `Handle`. The display returned is given by the box type and the number. This is similar to the method `GetGraphicItemByType(TypeName As String, ItemNumber As Integer) As Long`. The difference is the starting display box. `GetGraphicItemByType(TypeName As String, ItemNumber As Integer) As Long` starts at the top of the display while `GetSubgraphicItemByType` is relative to the display box referenced by handle.

GetTextOfGraphicItem(Handle As Long) As String

Copies the entire text of a graphic sub-item into a string. This will only work for relevant sub-items, like `TableBoxes`, `TextBoxes`, `NumberColBoxes`, and so forth.

JournalGraphicItem(Handle As Long) As Boolean

Copies the graphic sub-item to a JMP journal.

JournalOutput() As Boolean

Creates a journal for the current analysis window, without returning an automation object that allows further manipulation.

Launch() As Boolean

Runs the analysis with the columns that have been specified through LaunchAdd___.

LaunchAdd___(ColumnName As String) As Boolean

Adds the column name to the given role, where ___ should be X, Y, Freq, etc. (`LaunchAddX` for the X role, `LaunchAddFreq` for the Freq role, etc.). You must add columns before executing the `Launch` routine, otherwise the analysis will not know the source of the data.

LaunchAddBy() As Boolean

Adds the column name to be used for By Group processing. By default, when an analysis is launched, the functions specific to that analysis or graph will work on the first By Group result. Returns `True` if successful, `False` if unsuccessful.

LaunchRemove___(ColumnName As String) As Boolean

Removes the column name from the list of data sources for the impending analysis. ___ should be X, Y, Frequency etc., such as `LaunchRemoveX` for the X role, `LaunchRemoveFreq` for the Freq role, etc.

OutlineBoxGetTitle(Handle as Long) As String

Returns a string containing the title of the given outline box.

NumberColGetHeading(Handle As Integer)

Returns the heading text of a number column display box, providing a handle to the display box.

NumberColGetItemText(Handle As Long, ElementNumber As Integer) As String

For `NumberColBoxes` only. This retrieves the *i*th element of the NumberColBox, where *i* is determined by the second parameter. The number is returned as a string, and must be converted to a numeric to be used in numeric operations.

NumberColHide, StringColHide(Handle As Integer, Flag As Boolean)

Shows or hides the number column display box or string column display box.

NumberColSetHeading, StringColSetHeading(Handle As Integer, Title As String)

Sets the heading text of a number column display box or string column display box, providing a handle to the box and the text.

OutlineBoxSetTitle(Handle As Long, Title As String)

Sets the title of an OutlineBox identified by `Handle`. The OutlineBox handle must have been previously obtained using `GetGraphicItemByType`.

PrintPages(From As Integer, To As Integer) As Boolean

Prints the report, but only the given page range.

PrintReport() As Boolean

Prints the entire analysis report.

SaveGraphicItem(Handle As Long, FileName As String, GraphicType As Integer)

Saves the graphic sub-item to disk using the given filename. The format of the graphic is determined by the `GraphicType` parameter. This number should be obtained from the `jmpGraphicsFormats` constants. Examples are JPEG, PNG, SVG, TIFF, and Windows metafile.

SaveGraphicOutputAs(FileName As String, GraphicFormat As Integer)

Saves the entire analysis output to a file, using the `FileName` supplied. The graphic format should be obtained from the `jmpGraphicsFormats` constants. Examples are JPEG, PNG, SVG, TIFF, and Windows metafile.

SaveJournalAs(FileName As String) As Boolean

Generates a journal file for the report, and saves it to disk with the given `FileName`.

ScriptAction(JSLText As String)

Submits JSL to the analysis for interpretation.

SetFrameSize(X As Integer, Y As Integer)

Sets the size of the graph frame, as opposed to the entire analysis window. `X` and `Y` are in pixel coordinates.

SetPrintOrientation(printOrientConstants orientation) As Boolean

Sets the orientation of the printed output for the analysis to either `Landscape` or `Portrait`, based on the parameter that is passed in. Returns `True` for success, `False` for failure.

SetWindowPos(X As Integer, Y As Integer)

Sets the position of the analysis window relative to the Windows desktop. `X` and `Y` are in pixel coordinates.

SetWindowSize(CX As Integer, CY As Integer)

Sets the size of the analysis window in pixel coordinates.

StringColGetHeading(Handle As Integer)

Returns the heading text for the string column display box that the handle specifies.

StringColGetItemText(Handle As Long, ElementNumber As Integer) As String

For `StringColBoxes` only. This retrieves the *i*th element of the StringColBox, where *i* is determined by the second parameter. The item is returned as a string.

TableBoxMakeDataTable(Handle As Long) As DataTable

Creates a new data table and fills the cell values with the contents of the `TableBox` described by `Handle`.

UseByOutput(ByTitle As String) As Boolean

Allows you to access a By Group that is not first in the output order.

The string must be the title of the particular By Group that you want to manipulate. For example, suppose you have a Bivariate output that is grouped by sex with values “Female” and “Male”. If the “Female” reports are first in the output, they are the ones manipulated by the Bivariate automation functions. If you want to manipulate the “Male” reports, call the method

UseByOutput("sex=Male"). If the method returns True, subsequent Bivariate method calls operate on the “Male” output. To switch back to the “Female” output, issue a call UseByOutput("sex=Female").

DataFeed Object

The DataFeed object provides a way to read data from a serial port. This allows a user to hook up an instrument, read the values through JMP, and use them in the automation client program.

Methods

Close() As Boolean

Closes the connection to the port, cleans up the information about the port and tells JMP that this data feed object is closed. Any further attempt to use this data feed object will fail. A return value of True indicates success, False indicates Failure.

Connect(PortName As String) As Boolean

Attempts to establish a connection to the named port using the parameters that have either been previously specified in SetCommParms or in JMP. Returns True if successful, False otherwise. An example of a valid port name is COM1.

Disconnect() As Boolean

Disconnects from the port associated with this datafeed, but keeps the port information for use in a later Connect(PortName As String) As Boolean. A return value of True indicates success.

GetLine() As String

Reads a data element from a port and returns the value as a string.

SetCommParms(BSTR szCommPort, long baudrate, short parity, short databits, short stopbits, short flow) As Boolean

Set up the parameters for a serial port. If zero is specified for a particular value, then the values specified in JMP Preferences are used. If no preferences have been set, a default value is used. The szCommPort parameter is mandatory. Returns True if successful, False otherwise. An example of a valid port name is COM1. Values for parity should be obtained from the commParityConstants enumeration. Values for flow control can be a logical Or operation of any of the values from the commFlowControlConstants enumeration, or can be 0 for no flow control.

DataTable Object

The DataTable object exposes much of the functionality of the data table to automation. First, a DataTable dispatch pointer must be obtained by some other method, such as Document.GetDataTable or AUTODB.OpenTable.

Properties

Document

A property that returns the document object that contains the data table. This is valuable if you wish to run methods from the Document object class on the data table, such as `SaveAs(FileName As String)`.

NumberColumns

A property that shows the number of columns that the table contains. Can't be set.

NumberRows

A property that shows the number of rows in the data table. Can't be set.

Visible

A property that determines whether the data table is visible (`True`) or hidden (`False`). Can be set, as well as retrieved.

Methods

Activate() As Boolean

Brings the Data Table to the foreground and makes it the active table.

AddColumns(Prefix as String, NumToAdd As Integer, Where As Integer, Type As Integer,FieldWidth As Integer) As Integer

Adds `NumToAdd` columns to the data table, after the column specified by `Where` (e.g. Column 3). The type is provided with a constant from `colDataTypeConstants`, whose values are `Numeric`, `Character`, `RowState`, or `Unknown`. `FieldWidth` is only used for type `Character`. The function returns the number of columns successfully added.

AddNumericTableVar(Name As String, Value As Double)

Adds a numeric table variable.

AddRows(NumberToAdd As Integer, AddAfter As Integer)

Adds `NumberToAdd` rows after the row specified in `AddAfter`. Returns the number of rows successfully added.

If `AddAfter` is 0, the rows will be added to the top of the data table. If `AddAfter` is -1, or a number greater than the current number of rows in the table, the rows will be appended to the bottom of the table.

Note that `NumberToAdd` is a short integer, which means you can add only 32,767 rows at a time. A work-around is to add a single row after a large non-existent row. JMP adds that row after first creating all the rows needed between the last row in the data table and the new row. For example, the following line creates an empty data table 1,000,000 rows long by adding a single row after the 999,999th row.

```
dt.AddRows(1, 999999);
```

AddRowsHuge(NumberOfRows as Integer, AddAfterRow as Integer) As Integer

Adds large numbers (up to 2 billion) of rows. This method is an alternative to `AddRows`, which allows only 32,767 rows to be added at one time.

Adds `NumberOfRows` rows after row `AddAfterRow`. Returns the number of rows successfully added.

If `AddAfterRow` is 0, the rows will be added to the top of the data table. If `AddAfterRow` is -1, or a number greater than the current number of rows in the table, the rows are appended to the bottom of the table.

SummaryUnlinked() As Datatable

Similar to `Summary`, except the summary table that is created is not linked to the original table. While this means that brushing in one table does not affect the other, it also means that closing the original table does not close the Summary table.

AddStringTableVar(Name As String, Value As String)

Adds a string table variable.

AddToConcatList(ColumnName As String) As Boolean

Add a column to the list of columns to concatenate using the `Concatenate() As DataTable` method.

AddToJoinList(ColumnName As String)

Add the column as one that will participate in a `Join` operation.

AddToJoinMatchList(ColumnName As String) As Boolean

Adds a column to the list of those columns that will be used in a Matched Column Join. If the type is not Matched Column, then these entries are ignored. Each participating data table in a Matched Column Join should specify the columns that will be used for the match operation.

AddToSortList(ColumnName As String, Ascending As Boolean) As Boolean

Adds the column to the list of columns that determine how rows in the table will be sorted by a Sort method call. The first column that is specified is the main sorting column. Subsequent columns determine sorting within subgroups. If the Ascending flag is `True`, the sorting is done in ascending order. If it is `false`, it is done in descending order.

AddToSplitGroupList(ColumnName As String) As Boolean

Optional, this is a column whose values can uniquely identify each row in the new table.

AddToSplitList(ColumnName As String) As Boolean

Adds the column to the list of columns whose values are to form multiple new columns.

AddToStackList(ColumnName As String) As Boolean

Adds the column to the list of columns whose values will be "stacked" into a new column.

AddToSubList(ColumnName As String) As Boolean

Adds the column to the list of columns that will be used for the `Subset() As DataTable` command.

AddToSummaryGroup(ColumnName As String) As Boolean

Adds the column to the group list for the `Summary() As DataTable` operation.

AddToSummaryStatList(Stat As Integer)

Adds the statistic as one that should be performed during the `Summary() As DataTable` operation. The integer value for a statistic is obtained from the `dtSummaryStatConstants` definition.

AddToSummarySubGroup(ColumnName As String) As Boolean

Adds the column to the Sub Group list for `Summary() As DataTable` operation.

AddToTransposeList(Name as String) As Boolean

Adds a column name to the list of columns that will be used during a transpose of the data table. This simulates the behavior of selecting columns within the Transpose dialog and then clicking Add.

AddToTransposeByList(Name as String) As Boolean

Adds a column name to the list of columns that will be used as the grouping columns during a transpose of the data table. This simulates the behavior of selecting columns within the Transpose dialog then clicking By.

AddToUpdateMatchList(ColumnName as String) As Boolean

For the new `UpdateTable(DataTable2 as DataTable, IgnoreMissingValues As Boolean)` method, add a column for matching column operation. There must be another column added to the second data table using this same method, similar to the operation of `AddToJoinMatchList(ColumnName As String) As Boolean`.

ClearRowsSelection()

Clears the current rows selection.

ClearSelectedRowStates() As Boolean

Clears the Row States of the rows that have been selected using `SelectRows(StartRow As Integer, EndRow As Integer)`.

ColorByColumn(Name as String) As Boolean

Sets the row state color based on the values in the column specified by `Name`.

Concatenate() As DataTable

Concatenate the columns specified through `AddToConcatList(ColumnName As String) As Boolean`. Returns a dispatch pointer to the newly created data table, which can then be automated as well.

DeleteColumn(ColumnName As String)

Deletes the specified column from the table.

DeleteSelectedRows() As Boolean

Deletes the rows selected using the `SelectRows(StartRow As Integer, EndRow As Integer)` method. A return value of `True` indicates success.

Document() As Document

Returns the document object that contains the data table object.

CheckRowState(Index As Integer, rowStateConstants stateToCheck) As Boolean

This method provides a way to check the state of just one row. It does not build an enumerated list, so it can be simpler and more efficient to use this method if you only care about a specific row or rows. For example, if you want to see if Row 5 of a data table is hidden, call `CheckRowState(5, rowStateHidden)`. A return value of `True` indicates the row is hidden, `False` indicates that it is not.

EnumRowStatesBegin(rowStateConstants stateToCheck) As Integer

This builds a list, accurate at the time that this method is called, that contains the rows that have a specific Row State set. For example, if the `stateToCheck` parameter is `rowStateSelected`, then a list of the row numbers of all selected rows will be created. This method returns a number that indicates the total number of rows of the given state that are in the list.

This method works with these methods:

- `EnumRowStatesGetNextRow() As Integer`
- `EnumRowStatesGetRowByIndex(Index as Integer) As Integer`

It is essential to call `EnumRowStatesBegin(rowStateConstants stateToCheck) As Integer` before using the other two methods.

EnumRowStatesGetNextRow() As Integer

This method is used after a call to `EnumRowStatesBegin`. It returns a row number from the list created in `EnumRowStatesBegin`. Each call to this method returns the next row in this list, until the list has been fully traversed. For example, suppose `EnumRowStatesBegin(rowStateSelected)` builds a list that shows that rows 1, 4, and 7 are selected. Three consecutive calls to `EnumRowStatesGetNextRow` would return 1, 4 and 7 respectively.

A fourth call would return 0, since the list had been fully traversed.

This method works with these methods:

- `EnumRowStatesBegin(rowStateConstants stateToCheck) As Integer`

- `EnumRowStatesGetRowByIndex(Index as Integer) As Integer`

It is essential to call `EnumRowStatesBegin(rowStateConstants stateToCheck) As Integer` before using the other two methods.

EnumRowStatesGetRowByIndex(Index as Integer) As Integer

This method is used after a call to `EnumRowStatesBegin`. It returns a specific entry in the list that is built by `EnumRowStatesBegin`. For example, suppose `EnumRowStatesBegin(rowStateSelected)` builds a list that shows that rows 1, 4 and 7 are selected. A call to `EnumRowStatesGetRowByIndex`(2) would return a 4, for row 4.

Notice that this method does *not* return the status of row 2, but the second item of the enumerated list.

This method works with these methods:

- `EnumRowStatesBegin(rowStateConstants stateToCheck) As Integer`
- `EnumRowStatesGetNextRow() As Integer`

It is essential to call `EnumRowStatesBegin(rowStateConstants stateToCheck) As Integer` before using the other two methods.

ExcludeSelectedRows() As Boolean

Excludes the rows that have been selected using `SelectRows(StartRow As Integer, EndRow As Integer)` from subsequent calculations.

GetChangedRowStateVector(RowStateToCheck As RowStateConstants)

Returns an array of changed indices based on a certain state (such as selected or hidden). `RowStateBeginMonitoring` must be called first.

GetColumn(ColumnName As String) As Column

Retrieves a dispatch pointer to a table column, which can then be used to manipulate the column object through automation.

GetColumnByIndex(Index As Integer) As Column

Retrieves a dispatch pointer to a table column specified by the index (1, 2, 3, …).

GetColumnName(Index As Integer) As String

Returns the name of the column at Index as a string.

GetJSLFunctionErrorString As String

This functions just like the `GetRunCommandErrorString() As String` and `HasRunCommandErrorString() As Boolean` methods in the Application object. The `JSLFunction` method allows retrieval of JSL return values for successful calls.

See also `HasJSLFunctionErrorString As Boolean`.

An example of Visual Basic code to invoke these methods is:

```
DT.JSLFunction(Distribution(Columns(Height));
if (DT.HasJSLFunctionErrorString) Then
      MsgBox(DT.GetJSLFunctionErrorString)
Endif
```

GetNumberOfRowsByRowState(rowStateConstants stateToCheck) As Long

Returns the number of rows that are excluded, hidden, or selected, depending on which state is specified in the input parameter.

GetRowStatesChanged() As Boolean

This method returns a Boolean flag that shows if there have been any changes to the RowStates of rows within the data table since the last call to `GetRowStatesChanged()` or the initial opening of the data table. A return value of `True` indicates that the Row States have changed somewhere, a `False` means that they have not changed. This flag will be set to true if Selection, Hiding, Exclusion or Labeling states change.

GetRowStateVector

Returns an array of indices of row elements that meet a specific rowstate criterion (selected, hidden, excluded, labeled). This allows a caller to quickly get information on rowstates, and to see what has changed since the last time a query was made. The method declaration looks like:

```
GetRowStateVector(rowStateConstants state) As Variant
```

See also `GetRowStateVectorData`.

An example of using `GetRowStateVector` in Visual Basic:

```
myarray = DT.GetRowStateVector(rowStateSelected)'Get the vector of
selected row numbers
                        'Remember, JMP is 1 based for row numbers
nElements = UBound(myarray)   'Get the upper bound of the returned array
Dim Col As JMP.Column
Set Col = DT.GetColumn("name")

For i = 0 To nElements        'Display the names from the selected rows
      MsgBox (Col.GetCellVal(myarray(i)))
Next i
```

HasJSLFunctionErrorString As Boolean

This functions just like the `GetRunCommandErrorString` and `HasRunCommandErrorString` methods. The `JSLFunction` method allows retrieval of JSL return values for successful calls.

See also `GetJSLFunctionErrorString As String`.

An example of Visual Basic code to invoke these methods is:

```
DT.JSLFunction(Distribution(Columns(Height));
if (DT.HasJSLFunctionErrorString) Then
      MsgBox(DT.GetJSLFunctionErrorString)
```

```
Endif
```

HideSelectedRows() As Boolean

Hides the rows that have been selected using `SelectRows(StartRow As Integer, EndRow As Integer)`.

Join(DataTable2 As DataTable, JoinType As Integer, OutputTableName As String) As DataTable

Joins the rows that are specified through `AddToJoinList` from each table into a new table. A dispatch pointer to the new table is returned. `JoinType` is either by row, through Cartesian join, or by Matching Columns and is specified using one of the `dtJoinConstants`. `DataTable2` is the dispatch pointer of the second of the two tables participating in the join. Only the general options for the data table that calls Join will be used for the Join, i.e. the general options for `DataTable2` will be ignored.

If the Join is by Matching Columns, the Matching Column 0ptions that were set for both tables in `SetJoinMatchOptions` will be used. If no options are set, then a default operation is performed. The columns that were specified in `AddToJoinMatchList` are used to perform the match.

LabelSelectedRows() As Boolean

Labels the rows that have been selected using `SelectRows(StartRow As Integer, EndRow As Integer)`.

MarkerByColumn(Name as String) As Boolean

Sets the row state marker based on the values in the column specified by `Name`.

NewColumn(Name As String, Type As Integer, Model As Integer, Width As Integer)

Adds a new column with the specified name and type. `Type` is specified using one of the `colDataTypeConstants`. `Model` (`Continuous`, `Nominal`) is specified using one of the `colModelTypeConstants`.

PrintTable() As Boolean

Prints the table on the default printer.

ReorderColumns(ReorderType As Integer)

Reorders the columns either by data type, modeling, name or by reversing the order. The original order can also be restored. A value from `colReorderConstants` that is passed as the parameter defines the behavior.

RowStateBeingMonitoring

Begins keeping track of Row State changes so that an Automation function can return the information.

SelectColumn(Column As String, SelectFlag as Boolean) As Boolean

Selects the column whose name is provided if `SelectFlag` is `True`, otherwise de-select it. `False` is returned if the column is not found.

SelectExcludedRows() As Boolean

Selects the rows that are excluded. Returns `true` for success, `false` for failure.

SelectHiddenRows() As Boolean

Selects the rows that are hidden. Returns `true` for success, `false` for failure.

SelectLabeledRows() As Boolean

Selects the rows that are labeled. Returns `true` for success, `false` for failure.

SelectAllMatchingCells() As Boolean

Selects the cells that match the already selected row/column combinations.

For example, if a cell with value 58 is selected in the column "age", this method selects other cells with age equal to 58. Columns must be selected using the `SelectColumn` method on the DataTable object, or on a column object before calling one of these methods. `SelectAllMatchingCells` applies to all open data tables.

SelectMachingCells() As Boolean

Selects the cells that match the already selected row/column combinations.

For example, if a cell with value 58 is selected in the column "age", this method selects other cells with age equal to 58. Columns must be selected using the `SelectColumn` method on the DataTable object, or on a column object before calling one of these methods. `SelectMatchingCells` applies to the current data table.

SelectRandomly(SampleRate As Long) As Boolean

Randomly selects rows from the data table. If the value of `SampleRate` is greater than `1`, then `SampleRate` represents the number of rows that will be selected. If `SampleRate` is between `0` and `1`, that proportion of the data table will be selected.

SelectRows(StartRow As Integer, EndRow As Integer)

Selects the rows for an operation, as if they were highlighted using the mouse.

SelectRowsWhere(ColumnName As String, Operation As Integer, SelectHow As Integer,Comparative As String) As Boolean

Performs a `Select Where` operation, which mimics the functionality on the Rows menu. The column specified by ColumnName is compared using an operator defined in the `rowSelectWhereOperations` constants. These contain operations like `equals`, `greater than`, `less than`, and so forth. The `SelectHow` parameter determines how prior selected rows are treated. The new selection can either clear a previous selection, extend a previous selection, or be derived from the previous selection. The `rowSelectWhereHow` constants are used for this parameter.

Finally, the Comparative parameter defines what the operation is performed against. This is a string that contain a name, number or any value that is used to create the selection subset.

SetJoinMatchOptions(DropMultiples As Boolean, IncludeNonMatches As Boolean)

Sets the options for a Matching Column Join operation. Each participating data table can set these options, and both sets of options are honored in the Join. These options mirror the options of the Match Columns dialog. The default options are as follows: `DropMultiples` is False and `IncludeNonMatches` is False.

SetJoinMergeColumns(Boolean)

Turns on the option (`True`) to join like named columns, or to not merge them (`False`).

SetJoinOptions(UpdateFirstTable As Boolean, CopyFormulas As Boolean, SuppressFormulaEval As Boolean)

Sets the general options for the Join. All of the different types of Joins honor these settings. They mirror the options in the Join dialog. If a Matching Column join is performed, the Join will use the general options that belong to the calling data table. The default options are as follows: `UpdateFirstTable` is False, `CopyFormulas` is True and `SuppressFormulaEval` is True.

SetStackMultipleSeriesN(short N) As Boolean

When specified, this enables the multiple series stack, with N being the Number of Series. This must be done before Stack is called on the DataTable object. Returns `True` for success and `False` for failure.

SetTransposeOptions(OutputTableName as String, UseSelectedRows as Boolean) As Boolean

Sets the name of the output table created during a transpose and whether or not only the selected rows should be used or all the rows.

SetWindowPos(X As Integer, Y As Integer)

Sets the table window position relative to the Windows desktop. X and Y are in pixel coordinates.

SetWindowSize(CX As Integer, CY As Integer)

Sets the size of the table window in pixel coordinates.

Sort(Replace As Boolean) As DataTable

Sorts the table using the columns given to `AddToSortList(ColumnName As String, Ascending As Boolean) As Boolean`. If `Replace` is `True`, the existing table is rearranged with the sorted data and the pointer to the existing table is returned. If `Replace` is `False`, a new table is created with the sorted data and the dispatch pointer to the new table is returned.

Split(ColumnID As String, OutputTableName As String, KeepRemainingCols As Boolean)

Splits the table, using the `ColumnID` column to identify the new column names, the columns entered using `AddToSplitList(ColumnName As String) As Boolean` as the data, and the column entered in `AddToSplitGroupList(ColumnName As String) As Boolean` as the row identifier list.

Stack(idColumnName As String, stackedColumnName As String, TableName As String) As DataTable

Stacks the values from the columns specified in `AddToStackList(ColumnName As String) As Boolean`, using `idColumnName` to identify each row in the new table. The new stacked column is given the `stackedColumnName` in the new data table that is given the name specified in `TableName`. A dispatch pointer to the newly created data table is returned.

Subset() As DataTable

Takes the data that is a combination of the columns selected by `AddToSubList(ColumnName As String) As Boolean` and rows selected by SelectRows and creates a new data table with these values. If no columns had been added with `AddToSubList(ColumnName As String) As Boolean`, then all the columns are used in the subset.

SubsetSetRandomSelection(SampleRateOrSize as Double, Shuffle As Boolean) As Boolean

If `SampleRateOrSize` is greater than 0 and less than 1, it is treated as a Rate. If it is larger than 1, it is treated as the sample size. If `Shuffle` is set to 1, `SampleRateOrSize` is ignored and all the rows are shuffled in the table that is produced.

SubsetStratifyAddColumn(Column As String) As Boolean

Adds a table column to be used to stratify the random selection subset. Several columns can be added. After subset is called, the list of columns is emptied. You will need to specify the columns again if you perform another subset.

Summary() As DataTable

Creates a summary table using `AddToSummaryGroup(ColumnName As String) As Boolean`, `AddToSummarySubGroup(ColumnName As String) As Boolean` and `AddToSummaryStatList(Stat As Integer)`. A dispatch pointer to the newly created table is returned.

SummarySetStatColumnFormat(summaryStatColNameConstants format)

Sets the column name format for the columns produced by summary statistics.

Transpose() As DataTable

Does a simple transpose of the active data table, and returns a dispatch pointer to the newly created data table.

UpdateTable(DataTable2 as DataTable, IgnoreMissingValues As Boolean)

This mimics the behavior of the **Tables > Update** operation, where a table can be updated/merged with changed values from a second table. Matching Column operation is supported using the `AddToUpdateMatchList(ColumnName as String) As Boolean` method. If no Matching Columns are

added through the `AddToUpdateMatchList` method, than a normal Update is performed between the two tables.

A return value of `True` indicates success, `False` indicates failure.

Document Object

The Document object provides properties and methods that are common to the documents underlying each analysis or data table.

Properties

Application

Property containing a dispatch pointer to the Application object. Can not be set.

AutoSave

Property to specify if a save should be done automatically before documents that contain data tables are closed. True means save on close. Can be set or retrieved.

FullName

Property containing the full name of the document as a string. For retrieval only.

Name

Property containing the short name of the document as a string. For retrieval only.

Path

Property containing the full path of the document as a string. For retrieval only.

Saved

Property that indicates if a document has been modified since its last save. If `False`, the document has changes that haven't been saved. If `True`, there are no changes that are unsaved.

Visible

Property that determines if the document is visible (`True`) or hidden (`False`). Can be set and retrieved.

Methods

Activate()

Brings the window that contains the document to the foreground.

Close(SaveChanges as Boolean, FileName As String)

Closes the document. `SaveChanges` gives the user the option of specifying a different filename for a final save of the document. If `SaveChanges` is true, set `FileName` to the path and name of the file where you want the document saved. If you want to save to the existing document, it is recommended that the Save method be used.

CopyToClipboard()

Copies the contents of the document's window to the clipboard. If you are copying the contents of an analysis, it is strongly recommended that you use the CopyToClipboard method that each analysis implements.

CreateBivariate()

These methods create an analysis object of a specific type, e.g. Bivariate. A dispatch pointer to this object is returned to the caller. This object can then be used to specify columns for an analysis, launch the analysis, and then manipulate the analysis output. This method must be called before a particular analysis can be launched.

CreateCluster()

These methods create an analysis object of a specific type, e.g. Cluster. A dispatch pointer to this object is returned to the caller. This object can then be used to specify columns for an analysis, launch the analysis, and then manipulate the analysis output. This method must be called before a particular analysis can be launched.

CreateNormalMixtures() As NormalMixtures

This method creates an automation object for using the Normal Mixtures platform within JMP. This method replaces the use of KMNormalMixtures in the Cluster platform.

Create*Platform*()

These methods create an analysis object of a specific type, e.g. Bivariate. A dispatch pointer to this object is returned to the caller. This object can then be used to specify columns for an analysis, launch the analysis, and then manipulate the analysis output. This method must be called before a particular analysis can be launched.

Save()

If the document contains a data table, the data table is saved to disk. If it does not contain

a data table, nothing is done. There are methods for saving analysis output and journals

that are specific to those objects.

SaveAs(FileName As String)

If the document contains a data table, the data table is saved to disk using the filename that is provided. If it does not contain a data table, nothing is done. There are methods for saving analysis output and journals that are specific to those objects.

Journal

Using the Journal methods, you can save a journal as HTML, RTF, MS Word, or as a JMP Journal file.

Methods

GetActiveJournal() As Journal

Returns a handle to the current active journal to allow further manipulation.

SaveAsHTML(BSTR filename, jmpGraphicsFormats graphicType) As Boolean

Saves the journal as HTML to disk using the given filename. The second parameter indicates the type of graphics format that should be used for any pictures within the HTML output (e.g. `PNG`, `JPEG` or `METAFILE`).

SaveAsJournal(BSTR filename) As Boolean

Saves the journal to disk using the given filename. This can then be reloaded into a future JMP session.

SaveAsRTF(BSTR filename, jmpGraphicsFormats graphicType) As Boolean

Saves the journal as RTF (Rich Text Format) to disk using the given filename. The second parameter indicates the type of graphics format that should be used for any pictures within the RTF output (e.g. `PNG`, `JPEG` or `METAFILE`).

SaveAsMSWordDoc(Filename As String) As Boolean

Saves RTF-style output from the Journal as a Microsoft Word document with the given name. This works only if MS Word 2000 or above is installed on the client machine. Returns `True` if successful, `False` if it fails.

Text Import Object

The `TextImport` object provides a way to open a text file as a JMP data table, enabling you to specify the types of field and line delimiters that the text file uses. A pointer to the data table created from the import is returned to the caller. Before these methods can be used, a `TextImport` object must first be created. This can be done by calling `CreateTextImportObject(FileName As String, NumberColumns as Integer) As TextImport` on the Application object.

Methods

TextImport object methods provide a way to open a text file as a JMP data table, enabling you to specify the types of field and line delimiters that the text file uses.

ColumnNamesStart(StartLine as Integer)

Specifies the starting line for column headers. This implies that the file has column headers, so a positive value here obviates the need for a call to `FirstLineIsData(False)`. The line that contains column names must come before the first line of data.

DataStarts(StartLine As Integer)

Specifies the starting line for the row data. If the number specified is 1, than it is implied that there are no column headers. A call to `FirstLineIsData(True)` is not necessary in this particular case.

FirstLineIsData(Flag As Boolean)

Indicates if the first line of the text file should be interpreted as data or as column headers. `True` means data, `False` means header.

OpenFile() As Document

Opens the text file, using the options specified in the preceding methods. A Document object pointer is returned. To retrieve a object pointer to the underlying data table, use the `GetDataTable` method on the document object.

SetColumnType(ColumnNumber As Integer, Type As Integer) As Boolean

Forces the column to be either character or numeric. Use the `colDataTypeConstants` definition for the second parameter, but `RowState` is not a valid type for this operation.

SetEndOfFieldOptions(Options As Integer)

Specifies which delimiters should be used for end-of-field. This can be an combination of the values defined by `jmpTIEndOfFieldConstants`. In Visual Basic, the `Or` operator can be used to combine the values.

SetEndOfLineOptions(Options As Integer)

Specifies which delimiters should be used for end-of-line. This can be an combination of the values defined by `jmpTIEndOfLineConstants`.In Visual Basic, the `Or` operator can be used to combine the values.

StripQuotes(Flag As Boolean)

Specifies whether quotes should be removed from data before insertion into the new data table. `True` means strip quotes, `False` means keep them.

Platform Methods

Each platform has methods that enable you to launch and manipulate that platform.

Attribute Chart Object Methods

The Attribute Chart object provides a way to launch and manipulate the Attribute Gauge Chart platform.

EffectivenessReport(Flag As Boolean)

Turns the option on (`True`) or off (`False`).

Bivariate Object Methods

The Bivariate object provides a way to launch and manipulate a Bivariate analysis.

DensityEllipses(Degree As Double)

Draws a density ellipse with the given degree of probability.

FitEachValue As Fit

Fits each value on the analysis. Returns a reference to the `Fit` object, which allows further manipulation through automation.

FitLine As Fit

Performs a linear fit on the analysis. Returns a reference to the `Fit` object, which allows further manipulation through automation.

FitLoess() As Fit

Performs a Fit Loess using default parameters. Returns a reference to the `Fit` object, which allows further manipulation through automation.

FitLoessWeightConstants(fitLoessWeightTricube, fitLoessWeightCosine, fitLoessWeightEpanechnikov, fitLoessWeightGaussian, fitLoessWeightCauchy)

Specifies the Weight function for the Kernel (Local) Smoother method.

FitLoessWithParms(fitLoessLambdaConstants Lambda, Alpha as Double, Robustness as Short)

Performs a Fit Loess using the specified parameters. `Lambda` is a constant between 0 and 2, typically either Linear or Quadratic. `Alpha` is a value between 0 and 1 inclusive. `Robustness` is a value between 0 and 4 inclusive.

Returns a reference to the `Fit` object, which allows further manipulation through automation.

FitMean As Fit

Fits a mean on the analysis. Returns a reference to the `Fit` object, which allows further manipulation through automation.

FitOrthogonal(OrthogonalFitConstant as Integer, VarianceRatio As Double) As Fit

Performs an orthogonal regression with the specified variance ratio. The first parameter should be one of the values from the `bivarFitTransformConstants`. Returns a reference to the `Fit` object, which allows further manipulation through automation.

FitPolynomial(Degree As Double) As Fit

Performs a polynomial fit with the specified degree (e.g. 3.0). Returns a reference to the `Fit` object, which allows further manipulation through automation.

FitRobust, FitCauchy(Flag As Boolean) As Fit

Performs a Robust or Cauchy fit and returns a Fit object. You can then customize this method further.

FitSpline(Degree As Double) As Fit

Performs a spline fit with the specified degree of stiffness (e.g. 100). Returns a reference to the `Fit` object, which allows further manipulation through automation.

FitTransformed(Xtransform As Integer, Ytransform as Integer, PolynomialDegree as Integer)

Perform a fit with X and Y transformation.

- The `Xtransform` and `Ytransform` values come from the `bivarFitTransformConstants`, and the polynomial degree (e.g. 3) is similar to `FitPolynomial(Degree As Double) As Fit`.

FitTransformedWithOptions(Xtransform As Integer, Ytransform as Integer, PolynomialDegree as Integer, CenteredPolynomial as Boolean, ConstrainIntercept as Boolean, InterceptValue as Double, ConstrainSlope As Boolean, SlopeValue as Double) As Fit

Perform a fit with X and Y transformation and/or constraints.

- The `Xtransform` and `Ytransform` values come from the `bivarFitTransformConstants`, and the polynomial degree (e.g. 3) is similar to `FitPolynomial(Degree As Double) As Fit`.
- `CenteredPolynomial` is either `True` or `False`, and must be specified. The default for normal JMP operation is `True`.
- `ConstrainIntercept` is a Boolean value that indicates if there will be a constraint on the intercept. This must be set to `True` if you wish to specify an intercept value for the next parameter. If `ConstrainIntercept` is `False`, `InterceptValue` is ignored.
- `ConstrainSlope` is a Boolean that must be `True` if you wish to specify a value for the slope constraint. If it is `False`, `SlopeValue` is ignored.

GroupBy(ColumnName As String) As Boolean

Group the analysis output by values from a specific column, whose name is provided. Returns `True` for success, `False` for failure.

HistogramBorders(Flag as Boolean)

Matches the UI option to turn Histogram borders on (`True`) or off (`False`).

KernelSmoother(Lambda As JMP.fitLoessLambdaConstants, Weight As JMP.fitLoessWeightConstants, Alpha As Double, Robustness As Short)

Performs a Fit Loess using the specified parameters. `Lambda` is a constant between 0 and 2, typically either Linear or Quadratic. `Alpha` is a value between 0 and 1 inclusive. `Robustness` is a value between 0 and 4 inclusive.

Returns a reference to the `Fit` object, which allows further manipulation through automation.

NonParDensity() As FitDensity

Performs a nonparametric density estimation, returning a `FitDensity` object reference that allows for further manipulation of the output. (See "FitDensity Object Methods" on page 127.)

ShowPoints(Flag as Boolean)

Shows plot points if set `True` (1), or hides them if `False` (0).

Bubble Plot Object Methods

The `Bubble Plot` object provides a way to launch and manipulate bubble plots.

AggregateSizeAsSum(Flag As Boolean)

Turn this option on (`True`) or off (`False`).

AggregateXAsSum(Flag As Boolean)

Turn this option on (`True`) or off (`False`).

AggregateYAsSum(Flag As Boolean)

Turn this option on (`True`) or off (`False`).

AllLabels(Flag As Boolean)

Turn All Labels on (`True`) or off (`False`).

BubbleSize(Size as Double)

Specify the size of the bubble circle as a double value. 0 is smallest.

BubbleSpeed(Speed as Double)

Specify the speed of the bubble during animation as a double value. 0 is slowest.

BubbleTimeIndex(Index as Double)

Specify the starting point of the animation as a 0-based index of the values used for Time. For instance, if there are 5 distinct values for the Time column, 1.0 would specify starting exactly at the second value.

CombineAll()

When two ID columns are specified and `SplitAll()` has already been run, `CombineAll()` recombines the smaller bubbles into their original bubble.

Filled(Flag As Boolean)

Turn Fill on (`True`) or off (`False`).

Go()

Run animation forwards, looping to the beginning when the end is reached.

LaunchAddColoring(Name as BSTR)

In addition to the usual X and Y values, Bubble Plot provides launch methods to specify column values, such as Coloring, that are specific to the platform.

LaunchAddID(Name as BSTR)

In addition to the usual X and Y values, Bubble Plot provides launch methods to specify column values, such as ID, that are specific to the platform.

LaunchAddSizes(Name as BSTR)

In addition to the usual X and Y values, Bubble Plot provides launch methods to specify column values, such as Sizes, that are specific to the platform.

LaunchAddTime(Name as BSTR)

In addition to the usual X and Y values, Bubble Plot provides launch methods to specify column values, such as Time, that are specific to the platform.

Prev()

Run animation backward one Time unit.

SelectableAcrossGaps(Flag As Boolean)

Turn Selectable Across Gaps on (`True`) or off (`False`). SelectableAcrossGaps will only be available if a Time value was specified prior to running the Launch method.

SplitAll()

When two ID columns are specified, separate the bubble defined by the first ID into its smaller constituents defined by the second ID.

Step()

Run animation forward one Time unit.

Stop()

Stop the animation.

Trails(Flag As Boolean)

Turn Trails on (`True`) or off (`False`).

Categorical Response Analysis Methods

The `Categorical Response Analysis` object provides a way to launch and manipulate the categorical platform.

AgreementStatistic(Flag as Boolean) As Boolean

Turn this option on (`Flag` is `True`) or off. The option is specific to certain responses, and may not work if used with the wrong response type.

CrosstabFormat(Flag as Boolean) As Boolean

Turn this option on (`Flag` is `True`) or off. These options are mutually exclusive, so turning on one will turn off the others:

- `CrosstabFormat`
- `CrosstabTransposed`
- `TableFormat`
- `TableTransposed`

CrosstabTransposed(Flag as Boolean) As Boolean

Turn this option on (`Flag` is `True`) or off. These options are mutually exclusive, so turning on one will turn off the others:

- `CrosstabFormat`
- `CrosstabTransposed`
- `TableFormat`
- `TableTransposed`

Frequencies(Flag as Boolean) As Boolean

Turn this option on (`Flag` is `True`) or off.

FrequencyChart(Flag as Boolean) As Boolean

Turn this option on (`Flag` is `True`) or off.

LaunchAddResponseRole(ResponseType as jmpCategoricalResponseRoles) As Boolean

Add a list of columns as response roles; for example, **Aligned Responses**. All the column names that have been accumulated by calling `LaunchAddToResponseList` will be assigned to the role specified, and the list of columns used for `LaunchAddToResponseList` will be cleared. Multiple roles can be assigned to an analysis, with different lists of columns. Only when `Launch` is called will the analysis be created.

LaunchAddToResponseList(ColumnName as String) As Boolean

Add a column to a list that will be used to add one type of response role. You can add several columns to a list by calling `LaunchAddToResponseList` several times, then have all the list elements added as a role by calling `LaunchAddResponseRole`.

Legend(Flag as Boolean) As Boolean

Turn this option on (`Flag` is `True`) or off.

RatePerCase(Flag as Boolean) As Boolean

Turn this option on (`Flag` is `True`) or off.

ShareChart(Flag as Boolean) As Boolean

Turn this option on (`Flag` is `True`) or off.

ShareOfResponses(Flag as Boolean) As Boolean

Turn this option on (`Flag` is `True`) or off.

TableFormat(Flag as Boolean) As Boolean

Turn this option on (`Flag` is `True`) or off. These options are mutually exclusive, so turning on one will turn off the others:

- `CrosstabFormat`
- `CrosstabTransposed`
- `TableFormat`
- `TableTransposed`

TableTransposed(Flag as Boolean) As Boolean

Turn this option on (`Flag` is `True`) or off. These options are mutually exclusive, so turning on one will turn off the others:

- `CrosstabFormat`
- `CrosstabTransposed`
- `TableFormat`
- `TableTransposed`

TestEachResponse(Flag as Boolean) As Boolean

Turn this option on (`Flag` is `True`) or off. The option is specific to certain responses, and may not work if used with the wrong response type.

TestResponseHomogeneity(Flag as Boolean) As Boolean

Turn this option on (`Flag` is `True`) or off. The option is specific to certain responses, and may not work if used with the wrong response type.

TransitionReport(Flag as Boolean) As Boolean

Turn this option on (`Flag` is `True`) or off. The option is specific to certain responses, and may not work if used with the wrong response type.

Cell Plot Object Methods

The `Cell Plot` object provides a way to launch and manipulate cell plots.

LaunchOptions(BOOL Scale, BOOL Center)

Sets the Scale and Centering options prior to launch. Similar to the Cell Plot launch dialog. By default, these are Off (`False`).

Legend(Flag As Boolean)

Show the legend (`True`) or hide it (`False`).

Chart Object Methods

The Chart object provides a way to launch and manipulate the Chart platform. The original Chart platform in JMP has been deprecated. Automation code that calls the Chart platform will now use the equivalent Graph Builder commands to generate output. There may be differences in appearance from prior versions.

ConnectPoints(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`).

LaunchAddY(ColumnName As String, Statistic as Short)

Specify the Y column values using this different launch method. The first parameter is the usual column name. The second parameter is a statistical operation from `chartStatConstants`. These statistics match those found in the chart launch dialog. If you don't want a statistic performed on the data, specify the `Data` stat operation.

Orientation(WhichWay As Short)

Specifies if you want vertical or horizontal orientation for Bar, Needle, Line or Point plots. Get the value for the parameter from the `chartOrientConstants`.

Overlay(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`).

OverlayColor(Color As Short)

Specify the value of the overlay and line colors from the `jmpColorConstants` values.

SeparateAxes(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`).

ShowPoints(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`).

SpecifyQuantilesVal(Quantiles as Double) As Boolean

If a column with the Quantiles statistic is added, this method can be used to specify the quantile value. The quantile value is 25.0 by default. This method should be called before the column is added using `LaunchAddY(ColumnName As String, Statistic as Short)`.

SpecifyType(ChartType as Short)

Specifies the type (Bar, Needle, Point, Line, or Pie) of chart that you want to display. Get the value for the parameter from the `chartChartTypeConstants`.

Cluster Object Methods

The Cluster object provides a way to launch and manipulate both Hierarchical and *K*-Means cluster analyses. See "Common Analysis Functions" on page 83 for most details on starting the analysis.

Notes:

- There are two important Cluster-specific launch methods:
  - `LaunchSpecifyKMeans(Flag As Boolean)`
  - `LaunchSpecifyDistanceFormula(FormulaType As Integer)`
- Hierarchical and *K*-means clustering also each have methods specific to them. See Hierarchical Cluster Methods and KMeans Cluster Methods.

ClusterCriterion, ClusterSummary, ConstellationPlot (Flag As Boolean)

Turn the option on (`True`) or off (`False`).

ColorClusters(Flag As Boolean)

An On(`True`)/Off(`False`) option that mirrors its non-automation counterpart.

KMNormalMixtures(Flag as Boolean)

Support for this has been removed from JMP Clustering. This option is no longer supported. Use the new Normal Mixtures automation platform instead. See "Normal Mixtures Methods" on page 149.

KMParallelCoordPlots(Flag as Boolean) As Boolean

Displays the parallel coordinate plots for a K-Means Cluster analysis.

KMSOMBandwidth(Bandwidth As Double)

Specifies the bandwidth for the self-organized map.

LaunchSpecifyDistanceFormula(FormulaType As Integer)

Specifies the distance formula to use when computing the clusters (e.g. Centroid, Ward etc.). The `FormulaType` parameter should be a value from `clusterDistanceConstants`.

LaunchSpecifyKMeans(Flag As Boolean)

Indicates whether a Hierarchical (`False`) or *K*-Means (`True`) analysis should be performed. Once the Launch method is called, some methods will only work if called for their particular platform. The methods specific to each platform are detailed below.

Legend(Flag As Boolean)

Show the legend (`True`) or hide it (`False`).

MarkClusters(Flag As Boolean)

An On(`True`)/Off(`False`) option that mirrors its non-automation counterpart.

NumberOfClusters(Number As Integer)

Specifies the number of clusters to form.

ParallelCoordPlots, ScatterPlotMatrix

Turn the option on (`True`).

SaveClusters()

Saves the cluster number of each row in a new data table column.

Contingency Object Methods

The Contingency object provides a way to launch and manipulate a Contingency table analysis.

Cochran(ColumnName As String) As Boolean

Performs a Cochran-Mantel-Haenszel test, taking the provided column for blocking.

Correspondence(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`).

Crosstabs(Flag As Boolean) As Crosstabs

Turns the Crosstabs option on (`True`) or off (`False`). This method returns a dispatch pointer to a `Crosstabs` object, which allows further manipulation. See “Crosstabs Object Methods” on page 116.

HorizontalMosaic(Flag as Boolean)

Displays a horizontal mosaic plot (`True`) or a vertical mosaic plot (`False`). To display a mosaic plot, call `MosaicPlot(Flag As Boolean)` as `True`.

MosaicPlot(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`). If you do not use `HorizontalMosaic(Flag as Boolean)` to set a horizontal or vertical display, a vertical mosaic plot is displayed by default.

NomAxisBooleanOption(Handle as Long, Action as Short, Flag As Boolean)

Controls a Boolean option for the display of the Nominal Axis. If the platform supports the option, then this method will either turn it on (`Flag` is `True`) or off (`Flag` is `False`). Examples of options are Rotated Tick Labels, Divider Bars and displaying a Lower Frame. The Rotated Tick Labels are supported only on the Oneway and Variability Chart platforms. Before this method can be called, a handle to the Nominal Axis display box must be obtained through a call to `GetGraphicItemByType(TypeName As String, ItemNumber As Integer) As Long`.

Tests(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`).

Contour Object Methods

The `Contour` object provides a way to launch and manipulate Contour plots.

FillAreas(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`).

GenerateGrid(HorizontalSize As Integer, VerticalSize As Integer) As DataTable

Creates a new JMP data table with the number of grid coordinates requested and contour values computed from linear interpolation. A dispatch pointer to the newly created data table is returned.

LabelContours(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`).

ReverseColors(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`).

SaveContours() As DataTable

Saves the contour coordinate data in a new data table, and returns a dispatch pointer to the data table object to allow it to be manipulated.

SaveTriangulation() As DataTable

Saves the triangulation coordinate data in a new data table, and returns a dispatch pointer to the data table object to allow it to be manipulated.

ShowBoundary(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`).

ShowContours(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`).

ShowDataPoints(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`).

ShowTriangulation(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`).

ContourProfiler Object Methods

The `ContourProfiler` object provides a way to launch the Contour Profiler, manipulate the output using the common analysis methods, and the methods specific to Contour Profiler.

ContourGrid(Low As Double, High As Double, IncrementcAs Double)

Creates a grid of contour values, after specifying the `Low` and `High` limits and the increment.

ContourGridWithResponse(low as Double, high as Double, increment as Double, responseColumn As String) As Boolean

This functions the same as `ContourGrid(Low As Double, High As Double, IncrementcAs Double)`, except that it allows a response column to be entered, rather than using a default column. Enter the name of the column in the last parameter.

SurfacePlot(Flag As Boolean)

Turns the surface plot on (True) or off (False).

ControlChart Object Methods

The `ControlChart` object provides a way to launch and manipulate a variety of control charts. `ControlChart` contains quite a few unique launch methods that differ from the common launch methods.

BoxChart(Flag As Boolean)

A display option that can be set (`True`) or reset (`False`).

CenterColor(Color As Integer)

The connect color and center line color can be set by using these methods along with a value from `jmpColorConstants`.

ConnectColor(Color As Integer)

The connect color and center line color can be set by using these methods along with a value from `jmpColorConstants`.

ConnectPoints(Flag As Boolean)

A display option that can be set (`True`) or reset (`False`).

ConnectThroughMissing(Flag As Boolean)

A display option that can be set (`True`) or reset (`False`).

ControlLimits(Flag As Boolean)

A display option that can be set (`True`) or reset (`False`).

LaunchAddPhase, LaunchRemovePhase(ColumnName As String)

Adds or removes a phase variable before creating the control chart.

LaunchAddProcess(ColumnName As String) As Boolean

Selects a column for charting. For variables charts, specify measurements as the process. For attributes charts, specify the defect count or defective proportion as the process.

LaunchAddSampleLabel(ColumnName As String) As Boolean

Selects a column whose values label the horizontal axis.

LaunchAddSampleUnitSize(ColumnName As String) As Boolean

Selects a column to identify the rows that define subgroup samples.

LaunchSetChartType(ChartType As Integer)

Select the chart type that you want from the `jmpControlChartConstants`. This should be the first method that is called following the creation of the `ControlChart` object by `CreateControlChart`.

LaunchSetConstantSampleSize(Flag As Boolean, SampleSize As Integer)

When the first parameter is `True`, this says that you want to use `SampleSize` as a grouping constant, rather than a sample variable from a column.

LaunchSetCStats(various parms as double) As Boolean

Adds known chart statistics prior to launch. The parameters mirror those of the Control Chart dialog for the chart type that you are creating.

LaunchSetCusumOptions(TwoSided As Boolean, DataUnits As Boolean)

For the Cumulative Sum chart type, allows the two-sided and data units options to be set (`True` means On).

LaunchSetCusumStats(various parms as double) As Boolean

Adds known chart statistics prior to launch. The parameters mirror those of the Control Chart dialog for the chart type that you are creating.

LaunchSetEWMAStats(various parms as double) As Boolean

Adds known chart statistics prior to launch. The parameters mirror those of the Control Chart dialog for the chart type that you are creating.

LaunchSetEWMAWeight(Weight As Double)

If the EWMA chart type is selected, this allows you to specify the weight.

LaunchSetIRChartParms(IndMeas As Boolean, MovingRange As Boolean, Range As Integer)

For the IR Chart type, this allows the Individual Measurements and Moving Range options to be set. If Moving Range is selected, then the span should be specified as an integer.

LaunchSetIRStats(various parms as double) As Boolean

Adds known chart statistics prior to launch. The parameters mirror those of the Control Chart dialog for the chart type that you are creating.

LaunchSetIRSummarizeParms(PreSummarize As Boolean, Mean As Boolean, StdDev As Boolean)

Perform pre-summary statistics on the IR charts if the first parameter is `True`. Specify On Group Means, On Group Standard Deviations, or both. If you select *`Presummarize`*`=True`, the Sample Size will be derived from the Sample Label column if it has been specified. If there is no Sample Label column, or `LaunchSetConstantSampleSize(Flag As Boolean, SampleSize As Integer)` has been called, the Sample Size will be a constant.

LaunchSetKSigmaAlphaH(KSigma As Boolean, alpha As Boolean, H As Boolean, value As Double, beta As Double)

Allows the `KSigma`, `Alpha`, `H` and `Beta` parameters to be set, with `True` meaning set. `Beta` is specified as a `double`, as is `H`. `Beta` and `H` are only valid when the Cusum chart type is used.

LaunchSetNPStats(various parms as double) As Boolean

Adds known chart statistics prior to launch. The parameters mirror those of the Control Chart dialog for the chart type that you are creating.

LaunchSetPresummarizeChartTypes(VARIANT_BOOL IndivGroupMeans, VARIANT_BOOL IndivGroupStdDev, VARIANT_BOOL MovingRangeGroupMeans, VARIANT_BOOL MovingRangeStdDev) As Boolean

Provides On (`True`) / Off (`False`) switches for the four sub-chart types that are available for Presummarized output.

LaunchSetPresummarizeStats(double sigma, double meanMeasureGroup, double meanMeasureStdDev, double meanMovingGroup, double meanMovingStdDev) As Boolean

Sets the statistics for the Presummarize control chart type. This follows the fields from the dialog.

LaunchSetPStats(various parms as double) As Boolean

Adds known chart statistics prior to launch. The parameters mirror those of the Control Chart dialog for the chart type that you are creating.

LaunchSetUStats(various parms as double) As Boolean

Adds known chart statistics prior to launch. The parameters mirror those of the Control Chart dialog for the chart type that you are creating.

LaunchSetUWMAMovingAvg(Average As Double)

For UWMA chart types, this allows you to set the moving average span.

LaunchSetUWMAStats(various parms as double) As Boolean

Adds known chart statistics prior to launch. The parameters mirror those of the Control Chart dialog for the chart type that you are creating.

LaunchSetVariableChartParms(Xbar As Boolean, R As Boolean, S As Boolean)

For Variable charts, this sets (`True`) or resets (`False`) the `Xbar`, `R` and `S` parameters.

LaunchSetVariableStats(various parms as double) As Boolean

Adds known chart statistics prior to launch. The parameters mirror those of the Control Chart dialog for the chart type that you are creating.

Needles(Flag As Boolean)

A display option that can be set (`True`) or reset (`False`).

SaveLimits() As Datatable

Saves the control limits into a new table. A dispatch pointer to the new data table is returned.

SetAlarm(`jmpControlChartAlarms` alarmType) As Boolean

Sets the Control Chart alarm script to be one of a selection of written or spoken warnings. The `jmpControlChartAlarms` constants dictate the type of alarm that will be invoked when a test indicates an out of bounds condition.

Note: If you use this method, you *cannot* use `SetCustomAlarmText(BOOL Speak, BSTR text) As Boolean`.

SetCustomAlarmText(BOOL Speak, BSTR text) As Boolean

Sets the alarm text to the string that is passed in for the second parameter, rather than using a standard message. The first parameter dictates whether the alarm is spoken (`True`) or written to the log (`False`).

Notes:If you use this method, you *cannot* use `SetAlarm(jmpControlChartAlarms alarmType) As Boolean`**.**

SetActiveChart(chartNumber as Integer) As Boolean

Control Chart now allows manipulation of chart displays other than the topmost one. Use this method to select a chart other than the topmost one as the active chart. The chart ordering is 1 based. Subsequent calls to automation display methods will work on the active chart.

ShowCenter(Flag As Boolean)

A display option that can be set (`True`) or reset (`False`).

ShowLineLegend(Flag As Boolean)

A display option that can be set (`True`) or reset (`False`).

ShowPoints(Flag As Boolean)

A display option that can be set (`True`) or reset (`False`).

ShowZones(Flag As Boolean)

A display option that can be set (`True`) or reset (`False`).

Test(TestNumber As Integer, Flag As Boolean)

Runs a test with the given number if the flag is `True`, resets it if `false`. The test must be applicable to the chart for this to work.

TestsAll(Flag As Boolean)

Runs all the tests on the chart, if they apply and the flag is `True`.

WestgardRule(jmpControlChartRules ruleNumber, VARIANT_BOOL flag) As Boolean

Turns the specified rule On (`True`) or Off (`False`).

Crosstabs Object Methods

The `Crosstabs` object provides a way to manipulate the crosstabs output from Contingency.

CellChiSquare(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`).

Col(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`).

Count(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`).

Deviation(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`).

Expected(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`).

Row(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`).

Total(Flag As Boolean)

Turns the display option On (`True`) or Off (`False`).

Diagram Object

The `Diagram` object provides a way to launch the Diagram charting tool. However, due to the interactive nature of Diagram the manipulation of the Diagram after launch must be done interactively.

Methods

There are no methods specific to the Diagram object.

Discriminant Object Methods

The Discriminant object provides a way to launch and manipulate a Discriminant analysis.

CanonicalOptions(discrimScoreOptions option, Flag As Boolean) As Boolean

Select a Canonical Plot option and then specify if the option should be turned On (`Flag` is True) or Off (False). Examples are "Show Biplot Rays" and "Show Normal 50% Contours".

SaveDiscrimMatrices

This method doesn't take any parameters.

ScatterplotMatrix()

Generates a scatterplot matrix in a separate window.

ScoreData(Flag As Boolean) As Boolean

Turns the option On (True) or Off (False).

ScoreOptions(discrimScoreOptions option, Flag As Boolean) As Boolean

Select an option and then specify if the option should be turned On (`Flag` is True) or Off (False). Examples are "Show Classification Counts" and "Select Uncertain Rows".

ScoreSelectUncertainRows(Value As Double) As Boolean

This Score option takes a value where you specify how much the points differ from 0 or 1.

ShowCanonicalPlot(Flag As Boolean) As Boolean

Turns the option On (True) or Off (False).

ShowGroupMeans(Flag As Boolean) As Boolean

Turns the option On (True) or Off (False).

ShowWithinCovariances(Flag As Boolean) As Boolean

Turns the option On (True) or Off (False).

SpecifyPriors(discrimPriorsOptions option)

Allows Equal Probabilities and Proportional to Occurrence priors specifications to be used.

StepwiseSetup

This method doesn't take any parameters. `StepwiseSetup` brings up a selection panel, but you *cannot* automate the items within the panel. So, invoking `StepwiseSetup` will require user interaction to continue the analysis. Please use it with care.

DistribFit Object Methods

The `DistribFit` object, produced from the `Distribution` object `FitDistribution` method, allows further manipulation of the fit output.

DensityCurve(Flag As Boolean)

These are display options that can be set (`True`) or reset (`False`).

GoodnessOfFit(Flag As Boolean)

These are display options that can be set (`True`) or reset (`False`).

QuantilePlot(Flag As Boolean)

These are display options that can be set (`True`) or reset (`False`).

QuantilePlotAction(`distributionFitQuantilePlotConstants` action, VARIANT_BOOL flag) As Boolean

Manipulates the Distribution Fit Quantile Plot. The type of action that is performed (e.g. Rotate or turning on Confidence Limits) is governed by the first parameter. The flag turns the option on (`True`) or off (`False`).

Quantiles(UpperLimit As Double, LowerLimit As Double, Target As Double)

Returns the unscaled and uncentered distribution specific upper and lower percentiles that you specify.

LabelCumPoints(Flag As Boolean)

Turns the option on (`True`) or off (`False`).

RemoveFit()

Removes the fit from the analysis. The object pointer is no longer valid after this call.

SaveDensityFormula()

Saves the density values into a new column of the data table.

SaveFittedQuantiles()

Saves the fitted quantile values into a new column of the data table.

SpecLimits(lower as Double, upper as Double, target as Double)

Displays the specification limits for a capability analysis.

Distribution Object Methods

The `Distribution` object provides a way to launch and manipulate distribution analyses.

BetaBinomialFit(Sample Size as Integer, Sample Column as String) As Fit

Perform a Beta Binomial Fit on an existing Distribution. If you enter a numeric value for sample size, you must enter an empty string ("") as the Sample Column name. As an alternative, you can enter the name of a column that contains the sample size as the second parameter. In that case, the first parameter is ignored. A Fit object is returned for further manipulation.

BinomialFit(Sample Size as Integer, Sample Column as String) As Fit

Perform a Binomial Fit on an existing Distribution. If you enter a numeric value for sample size, you must enter an empty string ("") as the Sample Column name. As an alternative, you can enter the name of a column that contains the sample size as the second parameter. In that case, the first parameter is ignored. A Fit object is returned for further manipulation.

CapabilityAnalysis(LowerLimit As Double, UpperLimit As Double, Target As Double, Sigma As Double)

Performs a capability analysis with the provided lower spec limit, upper spec limit, target value and sigma.

CDFPlot(Flag As Boolean)

Display option that can be set by specifying `True` for parameter, or reset by specifying `False`.

ConfidenceInterval(Alpha As Double)

Compute the confidence intervals, with the given alpha levels, for means and standard deviations if the columns are continuous and for proportions if the columns are discrete.

CountAxis(Flag As Boolean)

A display options for histograms, it can be turned on by specifying `True`, or off by specifying `False`.

DensityAxis(Flag As Boolean)

A display options for histograms, it can be turned on by specifying `True`, or off by specifying `False`.

ErrorBars(Flag As Boolean)

A display options for histograms, it can be turned on by specifying `True`, or off by specifying `False`.

FitDistribution(FitType As Integer) As FitDistribution

Performs one of several available Fits on the data, and returns a pointer to a `FitDistribution` object. This allows further manipulation of the fit output. `FitType` is one of the values in `fitDistribConstants`.

FitNormalMixtures(NumberOfClusters as Integer) As FitDistribution

Performs a Normal Mixtures fit, specifying the number of clusters. For a Normal 2 Mixture or Normal 3 Mixture fit, such as those that are available in the UI, specify 2 and 3 respectively for `NumberOfClusters`.

Histogram(Flag As Boolean)

A display options for histograms, it can be turned on by specifying `True`, or off by specifying `False`.

HorizontalLayout(Flag As Boolean)

Rotates the graphical output from a vertical to a horizontal orientation if the flag is set to `True`.

Other distribution methods are specific to analyses of continuous variables or nominal/ordinal variables.

Moments(Flag As Boolean)

Display option that can be set by specifying `True` for parameter, or reset by specifying `False`.

MoreMoments(Flag As Boolean)

Display option that can be set by specifying `True` for parameter, or reset by specifying `False`.

MosaicPlot(Flag As Boolean)

This method is specific to nominal or ordinal distributions. Displays the mosaic plot (`True`) or hides it (`False`).

NomAxisBooleanOption(Handle as Long, Action as Short, Flag As Boolean)

Controls a Boolean option for the display of the Nominal Axis. If the platform supports the option, then this method will either turn it on (`Flag` is `True`) or off (`Flag` is `False`). Examples of options are Rotated Tick Labels, Divider Bars and displaying a Lower Frame. The Rotated Tick Labels are supported only on the Oneway and Variability Chart platforms. Before this method can be called, a handle to the Nominal Axis display box must be obtained through a call to `GetGraphicItemByType(TypeName As String, ItemNumber As Integer) As Long`.

NormalQuantilePlot(Flag As Boolean)

Display options that can be set by specifying `True` for parameter, or reset by specifying `False`.

OutlierBoxPlot(Flag As Boolean)

Display option that can be set by specifying `True` for parameter, or reset by specifying `False`.

PredictionInterval(alpha as Double, nSamples as Long)

Displays the prediction interval. Note that this option can output the prediction interval for the mean of *n* samples.

ProbAxis(Flag As Boolean)

A display options for histograms, it can be turned on by specifying `True`, or off by specifying `False`.

QuantileBoxPlot(Flag As Boolean)

Display option that can be set by specifying `True` for parameter, or reset by specifying `False`.

Quantiles(Flag As Boolean)

Display option that can be set by specifying `True` for parameter, or reset by specifying `False`.

Save(Action As Integer)

Allows a variety of analysis results to be saved into the data table. The action should be a value from `distributionSaveConstants`.

SetQuantileIncrement(Increment As Double)

Sets the quantile increment if the distribution is based on continuous data. This method does affect nominal or ordinal data.

ShowCounts(flag as Boolean)

Displays the counts on the histogram.

ShowPercents(flag as Boolean)

Displays the percentages on the histogram.

StemAndLeaf(Flag As Boolean)

Display options that can be set by specifying `True` for parameter, or reset by specifying `False`.

TestMean(meanToTest As Double, Sigma As Double, Wilcoxon As Boolean)

Allows you to test a hypothetical value for statistical comparison to the mean.

TestMeanWithOptions(meanToTest As Double, Sigma As Double, Wilcoxon As Boolean, PValue As Boolean, Power As Boolean)

Adds the ability to do a Power or P-value animation to the test of mean. True for the P-value or Power parameters indicate that the animation should be done, False means don't do the animation.

TestStdDev(stdDeviation As Double)

Tests a hypothesized standard deviation against a sample standard deviation.

ToleranceInterval(Alpha as double, Proportion as double)

Provides the same feature as the Tolerance Interval option/dialog under the UI.

DOE Object Methods

Automation support for part of Design of Experiments (DOE) is included for the first time in JMP 6. The major features of Custom Design that are supported by JSL (JMP Scripting Language) are also supported by Automation. Before the DOE methods can be called, a DOECustom object must be created. This is done by calling `CreateDOECustom () As DOECustom` on the JMP Application Automation object.

Notes:Order is important with DOE automation. You should call the methods in the same order you would perform the operations when using DOE with a user interface. For example, make sure to call `SimulateResponses()` before creating a table. Make sure to call `MakeModel(ModelType As doeModelTypes) As Boolean` *before* calling `MakeDesign()`.

AddBlockingFactor(NumberOfRuns As Long) As Boolean

Add a Blocking factor, which requires you to specify the number of runs. Returns `True` for success, `False` for failure.

AddCategoricalFactorWithLevelNames(FactorName as String, LevelNames as Variant Array of Strings) As Boolean

Adds a Categorical factor, specifying the factor name and the name of each level within that factor. The level names must be specified in an array of strings. Depending on the automation client used, this might need to be declared as a Variant and then re-dimensioned as a String array, or just declared as a String array. It is highly suggested that you consult the DOE sample automation program provided with the JMP install to see how to use this method.

AddBlockingFactorWithName(FactorName As String, NumberOfRuns As Long) As Boolean

Adds a Blocking factor, this time specifying the name of the factor rather than using a default name. The number of runs must still be specified.

AddCategoricalFactor(NumberOfLevels as Long) As Boolean

Adds a Categorical factor. You must specify the number of levels.

AddCategoricalFactorWithName(FactorName As String, NumberOfLevels as Long) As Boolean

Adds a Categorical factor, specifying a name for the factor. You must specify the number of levels.

AddContinuousFactorWithBounds(LowerBound As Double, UpperBound As Double) As Boolean

Adds a Continuous factor, allowing you to specify the lower and upper bounds.

AddContinuousFactorWithName(FactorName as String, LowerBound As Double, UpperBound As Double) As Boolean

Adds a Continuous factor, specifying the factor name. You must specify the lower and upper bounds.

AddFactor(factorType As doeFactorType)

Using the `doeFactorType` constants, add a factor type (e.g. Continuous, Categorical, Mixed) using the default settings.

AddMixtureFactorWithBounds(LowerBound As Double, UpperBound As Double) As Boolean

Adds a Mixture factor, allowing you to specify the lower and upper bounds.

AddMixtureFactorWithName(FactorName as String, LowerBound As Double, UpperBound As Double) As Boolean

Adds a Mixture factor, specifying the factor name. You must specify the lower and upper bounds.

AddResponse(ResponseType as doeResponseTypes, Name as String, LowerLimit As Double, UpperLimit As Double, Importance As Double) As Boolean

Add a response to the design. This should be done before invoking `MakeModel(ModelType As doeModelTypes) As Boolean`, `MakeDesign()`, or `MakeTable() As Boolean`. The `doeResponseType` constants contain the goal types (Maximize, Minimize, etc.).

AddTerms(Terms as Variant Array) As Boolean

Add a product of factors to the terms for the model. The factors that are involved must have default names like X1, X2, X3. The array that is passed in to `AddTerms` defines a numeric list of the X factors that you would like to cross for the new term. For example, 1,2 and 3 would result in the term X1*X2*X3. The term numbers must be specified in an array of type Long. Depending on the automation client used, this might need to be declared as a Variant and then re-dimensioned as a Long array, or just declared as a Long array. It is highly suggested that you consult the DOE sample automation program provided with the JMP install to see how to use this method.

Notes:You should call `AddTerms` before invoking `MakeDesign()`.

AddTermsWithPowers(Terms as Variant Array, Powers as Variant Array) As Boolean

Add a product of factors to the terms for the model, with the factors having exponents. Each element in the Term array must have a matching exponent in the Powers array. So, to have the term X1*X2^2*X3^4 you would define the Terms array with elements 1, 2 and 3. You would define the Powers array with elements 1, 2, and 4.

Depending on the automation client used, these arrays might need to be declared as a Variant and then re-dimensioned as a Long array, or just declared as a Long array. It is highly suggested that you consult the DOE sample automation program provided with the JMP install to see how to use this method.

Notes:You should call `AddTermsWithPowers` **before invoking** `MakeDesign()`**.**

LoadResponses(Table as DataTable) As Boolean

This method can be used to load the design responses from an existing automation data table.

The table must already by loaded using `OpenDocument(FileName As String) As Document`, and the object passed back from `OpenDocument::GetDataTable` must be passed to this method. A return value of `True` indicates success, `False` indicates failure. This method should be called before invoking `MakeModel(ModelType As doeModelTypes) As Boolean`, `MakeDesign()`, and `MakeTable() As Boolean`.

Related methods are:

- `LoadFactors(Table as DataTable) As Boolean`
- `LoadConstraints(Table as DataTable) As Boolean`

LoadFactors(Table as DataTable) As Boolean

This method can be used to load the design factors from an existing automation data table.

The table must already by loaded using `OpenDocument(FileName As String) As Document`, and the object passed back from `OpenDocument::GetDataTable` must be passed to this method. A return value of `True` indicates success, `False` indicates failure. This method should be called before invoking `MakeModel(ModelType As doeModelTypes) As Boolean`, `MakeDesign()`, and `MakeTable() As Boolean`.

Related methods are:

- `LoadResponses(Table as DataTable) As Boolean`
- `LoadConstraints(Table as DataTable) As Boolean`

LoadConstraints(Table as DataTable) As Boolean

This method can be used to load the design constraints from an existing automation data table.

The table must already by loaded using `OpenDocument(FileName As String) As Document`, and the object passed back from `OpenDocument::GetDataTable` must be passed to this method. A return value of `True` indicates success, `False` indicates failure. This method should be called *before* invoking `MakeModel(ModelType As doeModelTypes) As Boolean`, `MakeDesign()`, and `MakeTable() As Boolean`.

Related methods are:

- `LoadResponses(Table as DataTable) As Boolean`
- `LoadFactors(Table as DataTable) As Boolean`

MakeDesign()

Make the design. Call this after adding factors, calling `MakeModel(ModelType As doeModelTypes) As Boolean`, and adding terms.

MakeModel(ModelType As doeModelTypes) As Boolean

Make the DOE model, using the model type constants like RSM, Linear, and Interactions. You should call this method *after* adding factors, but *before* calling `MakeDesign()` and `MakeTable() As Boolean`.

MakeTable() As Boolean

Produce the design table. If `SimulateResponses()` was called previously, then the table will contain simulated results as well as the completed design.

NumberOfCenterpoints(nCenterpoints As Long) As Boolean

Enter the number of center points if desired, *before* calling `MakeTable() As Boolean`.

NumberOfReplicates(nReplicates as Long) As Boolean

Enter the number of replicates if desired, *before* calling `MakeTable() As Boolean`.

NumberOfStarts(nStarts As Long) As Boolean

Enter a positive whole number to specify the number of random starting designs. Do this *before* calling `MakeDesign()`.

OptimalityCriterion(Criterion as doeOptimalityConstants) As Boolean

Specify an optimality other than Recommended *before* calling `MakeDesign()`. Choices are D-Optimal and I-Optimal.

SaveFactors()

Save the factors for the design to a new data table.

SaveXMatrix()

Save the design matrix as a Table variable in the final table output. This method functions as a toggle, the first time it is called SaveXMatrix will be activated. If it were called again, it would be turned off.

SetRandomSeed(Seed As Double) As Boolean

Enter a positive whole number if you wish to specify the seed for the random starting design. If -1 is entered for the seed, then a prompt is presented for the seed. This should be called before invoking `MakeDesign()`.

ShowDiagnostics()

Turn on diagnostics. This method functions as a toggle: the first time turning the feature on, the next time it is called it turns the feature off.

SimulateResponses()

Simulate responses for the final design table. This should be called before invoking `MakeTable() As Boolean`. This method is a toggle: the first time it is called it turns the feature on, the next time it is called it turns the feature off.

SpecifyChangeDifficulty(doeChangeDifficultyConstants difficulty)

Change the level of difficulty for factor modification (Easy, Hard) in DOE automation. Factors that are added use the following automation methods:

- `AddCategoricalFactorWithLevelNames(FactorName as String, LevelNames as Variant Array of Strings) As Boolean`
- `AddCategoricalFactorWithName(FactorName As String, NumberOfLevels as Long) As Boolean`
- `AddContinuousFactorWithName(FactorName as String, LowerBound As Double, UpperBound As Double) As Boolean`
- `AddMixtureFactorWithName(FactorName as String, LowerBound As Double, UpperBound As Double) As Boolean`

SphereRadius(Radius as Double)

Enter a positive number if desired, for the radius of the spherical design region.

Fit Object Methods

The `Fit` object allows further manipulation of a fit display. This object is returned from several Bivariate object methods.

ConfidenceFit(Flag As Boolean)

Turns on (1 or `True`) or off (0 or `False`) the options for confidence curves.

ConfidenceIndividual(Flag As Boolean)

Turns on (1 or `True`) or off (0 or `False`) the options for 95% confidence limits.

LineOfFit(Flag As Boolean)

Turns on (1 or `True`) or off (0 or `False`) the options for the line of fit.

PlotResiduals(Flag As Boolean)

Creates a plot of residual values if the parameter is `True`.

RemoveFit()

Removes the fit from the Bivariate display output.

SavePredicteds()

Creates a new column in the data table with predicted values of Y.

SaveResiduals()

Creates a new column in the data table with residual values of Y.

SetAlpha(Alpha As Double)

Sets the alpha value for the Fit.

SplineSaveCoeffs()

Saves the spline coefficients in the original data table. If you have a `Fit` object obtained from a Spline fit, you can use this method. It will return a data table object that can be manipulated further. See also `SplineSavePredFormula() As DataTable`.

SplineSavePredFormula() As DataTable

Saves the spline prediction formula in a new data table. If you have a `Fit` object obtained from a Spline fit, you can use this method. See also `SplineSaveCoeffs()`.

FitDensity Object Methods

The `FitDensity` object allows further manipulation of the Nonparametric Density output. (See `NonParDensity() As FitDensity`.)

FivePercentContours(Flag As Boolean)

An On(`True`)/Off(`False`) option that mirrors its non-automation counterpart.

KernelControl(Flag As Boolean)

An On(`True`)/Off(`False`) option that mirrors its non-automation counterpart.

MeshPlot(Flag As Boolean)

An On(`True`)/Off(`False`) option that mirrors its non-automation counterpart.

ModalClustering(Flag As Boolean)

An On(`True`)/Off(`False`) option that mirrors its non-automation counterpart.

SaveDensityGrid() As DataTable

Saves the density estimates and quantiles associated with them in a new data table. A dispatch pointer to this new data table is returned, so it can be automated as well.

FitLeastSquares Object Methods

These methods are returned from the call to launch when the fitting personality prior to the launch is `Standard Least Squares`. The `FitLeastSquares` object allows the profilers to be invoked on the Response output. It also allows a particular Response to be manipulated, by facilitating retrieval of a Response object.

ContourProfiler(Flag As Boolean) As Boolean

Turns the Contour Profiler on (`True`) or off (`False`). Returns `True` for success, `False` for failure.

CubePlot(Flag As Boolean) As Boolean

Turns the Cube Plot on (`True`) or off (`False`). Returns `True` for success, `False` for failure.

GetResponse(Name As String) As FitResponse

Returns a particular `Response` object associated with the Model output. There is a `Response` object for every Y value entered prior to Model launch.

Profiler(Flag As Boolean) As FitProfiler

Turns the Profiler on (`True`) or off (`False`). A `FitProfiler` object is returned that allows for further manipulation of the Profiler settings.

FitLogvariance Object Methods

The LogVariance Fit output has a few options that can be specified after launch.

ConfidenceInterval(Alpha As Double) As Boolean

Specifies the confidence interval.

LikelihoodRatio(Flag As Boolean) As Boolean

Turn the option on (`True`) or off (`False`).

MarginalVariances (Flag As Boolean) As Boolean

Turn the option on (`True`) or off (`False`).

FitManova Object Methods

The Manova fitting personality has very limited support, due to the highly interactive nature of the Response specification dialogs. However, there is support for saving values to the active data table.

SaveDiscrim() As Boolean

Saves the specified value to columns in the active data table.

SavePredicted() As Boolean

Saves the specified value to columns in the active data table.

SaveResiduals() As Boolean

Saves the specified value to columns in the active data table.

Fit Model Methods

The Fit Model methods are used to launch a Fit Model analysis, and then to manipulate the subsequent output. Because the post-launch manipulation is so interactive and specific to the data, the Manova fitting personality only has limited support in the post-launch phase.

Notes:

FitModel produces a variety of output objects, such as FitLeastSquares, FitManova, and so forth. Because of the complexity of the output, the common analysis routine UseByOutput(ByTitle As String) As Boolean cannot be used for FitModel when By Group manipulation must be performed.

The Launch methods are specific to the launch setup for a Fit Model analysis. In addition to these methods, certain standard launch functions serve a dual purpose with Fit Model. For the Proportional Hazards and Parametric Survival fitting platforms, the launch buttons **Time To Event** and **Censor** equate to the LaunchAddY and LaunchAddWeight automation routines, respectively.

These four methods are used to retrieve the names of effects created using the AddxxxEffect methods, remove selected effects, and Nest effects:

- LaunchGetNumberOfEffects() As Integer
- LaunchGetEffectName(EffectNumber As Integer) As String
- LaunchSelectEffect(EffectNumber As Integer, OnOffFlag As Boolean) As Boolean
- LaunchRemoveSelectedEffects() As Boolean

Launch() As Object

Launches the fit using all of the previously supplied information. Depending on the type of personality that was selected, another Fit object will be returned that allows for further manipulation of the post-launch output. The objects that can be returned are FitLeastSquares, FitStepwise, FitNominal, FitOrdinal, FitLogVariance, FitProportional, and FitParametricSurvival. There is no object for the Manova fitting personality, due to the highly interactive nature of its output.

LaunchAddCrossEffect() As Boolean

Creates a crossed model effect, using the columns specified in calls to LaunchAddToEffectList(Name As String) As Boolean.

LaunchAddMacroEffect(fitModelMacroEffectConstants macroType) As Boolean

Adds a macro effect type, using the columns previously specified through LaunchAddToEffectList(Name As String) As Boolean, the macro degree specified using LaunchSpecifyMacroDegree, and the type of macro effect passed in as the macroType parameter. The macro type is one of the fitModelMacroEffectConstants. If a macro degree has not been previously specified, degree 2 is used.

LaunchAddNestEffect() As Boolean

Enables the column that has been added using LaunchAddToEffectList(Name As String) As Boolean to be used as a nesting effect within the effect that has been selected using LaunchSelectEffect(EffectNumber As Integer, OnOffFlag As Boolean) As Boolean.

For example, suppose the column ID (the subject ID within each treatment) is added as a simple X Effect (LaunchAddToEffectList(Name As String) As Boolean followed by LaunchAddXEffect() As Boolean). Next, the column Dose is added using LaunchAddToEffectList(Name As String) As Boolean. Finally, LaunchAddNestEffect() As Boolean is called and it creates the effect ID[Dose].

LaunchAddToEffectList(Name As String) As Boolean

Adds the column specified by `Name` to the effects columns list. This is the list of columns used when creating Model Effects using the `LaunchAddXEffect`, `LaunchAddNestEffect`, `LaunchAddCrossEffect`, and `LaunchAddMacroEffect` methods. This list does not contain the Effects created by these methods.

LaunchAddXEffect() As Boolean

Creates a simple model effect. This is the same as using the **Add** button in the Fit Model launcher dialog. The columns currently in the effect column list are used. These must have been specified using `LaunchAddToEffectList(Name As String) As Boolean`.

LaunchAddXEffectWithTransform(transform as fitModelTransforms)

Add a transformed effect. Examples are `Exponential` and `Square`. The name of the column to use for the effect must have been added previously using `LaunchAddToEffectList(Name As String) As Boolean`.

LaunchAddYWithTransform(ColumnName As String, transform as fitModelTransforms) As Boolean

Add a Y variable to the analysis, supplying a transform like Log or Sqrt. Returns `True` for success, `False` for failure.

LaunchGetEffectName(EffectNumber As Integer) As String

Returns a string that identifies a particular effect. For example, `Height*Weight` is returned for a crossed effect using the columns `Height` and `Weight`.

This method and the following three are used to retrieve the names of effects created using the `AddxxxEffect` methods, remove selected effects, and Nest effects:

- `LaunchGetNumberOfEffects() As Integer`
- `LaunchSelectEffect(EffectNumber As Integer, OnOffFlag As Boolean) As Boolean`
- `LaunchRemoveSelectedEffects() As Boolean`

LaunchGetNumberOfEffects() As Integer

Retrieves the number of effects that have been created using the various `AddEffect` methods. This allows the caller to enumerate the effects.

This method and the following three are used to retrieve the names of effects created using the `AddxxxEffect` methods, remove selected effects, and Nest effects:

- `LaunchGetEffectName(EffectNumber As Integer) As String`
- `LaunchSelectEffect(EffectNumber As Integer, OnOffFlag As Boolean) As Boolean`
- `LaunchRemoveSelectedEffects() As Boolean`

LaunchRemoveFromEffectList(Name As String) As Boolean

Removes the column specified by `Name` from the effects columns list.

LaunchRemoveSelectedEffects() As Boolean

Removes all of the effects that are currently in the list created by calls to `LaunchSelectEffect(EffectNumber As Integer, OnOffFlag As Boolean) As Boolean`. These effects will not be used in the modeling calculations. After this method is invoked, the effect list is emptied.

This method and the following three are used to retrieve the names of effects created using the `AddxxxEffect` methods, remove selected effects, and Nest effects:

- `LaunchGetNumberOfEffects() As Integer`
- `LaunchGetEffectName(EffectNumber As Integer) As String`
- `LaunchSelectEffect(EffectNumber As Integer, OnOffFlag As Boolean) As Boolean`

LaunchSelectEffect(EffectNumber As Integer, OnOffFlag As Boolean) As Boolean

Adds the particular effect identified by the `EffectNumber` to an internal list of effects that can then be removed using `LaunchRemoveSelectedEffects() As Boolean`, or that can have attributes specified for them using `LaunchSpecifyAttributesForSelectedEffects(fitModelEffectAttributeConstants attribNumber) As Boolean`.

This method and the following three are used to retrieve the names of effects created using the `AddxxxEffect` methods, remove selected effects, and Nest effects:

- `LaunchGetNumberOfEffects() As Integer`
- `LaunchGetEffectName(EffectNumber As Integer) As String`
- `LaunchRemoveSelectedEffects() As Boolean`

LaunchSpecifyAttributesForSelectedEffects(fitModelEffectAttributeConstants attribNumber) As Boolean

Specifies attributes for the effects that have been selected using `LaunchSelectEffect(EffectNumber As Integer, OnOffFlag As Boolean) As Boolean`. This mirrors the **Attributes** popup menu from the Fit Model dialog. Examples of effects are `Mixture Effect` and `Random Effect`.

The effect type should be specified using one of the `fitModelEffectAttributeConstants`. All of the effects currently in the effect list are given this attribute. The effect list is then emptied.

LaunchSpecifyDistribution(`fitModelDistributionConstants`) As Boolean

Used to specify the distribution when the `Parametric Survival` fitting personality is selected. Possible choices are `Weibull`, `LogNormal`, and `Exponential`, and should be specified using

`fitModelDistributionConstants`. If `Parametric Survival` is not specified, this setting is ignored.

LaunchSpecifyEmphasis(`fitModelEmphasisConstants` emphasis) As Boolean

Used to specify the emphasis when the `Standard Least Squares` fitting personality is selected. This is equivalent to the drop-down list found in the Fit Model dialog. Possible choices are

Effect Leverage, Effect Screening, and Minimal Report. If Standard Least Squares is not selected, this setting is ignored.

LaunchSpecifyIntercept(Flag As Boolean)

Turns Intercept on (True) or off (False). By default, Intercept is turned off.

LaunchSpecifyPersonality(fitModelPersonalityConstants personality) As Boolean

Used to define the fitting personality for the analysis. Examples are Standard Least Squares, Loglinear Variance and Parametric Survival. Standard Least Squares is the default personality.

Some personalities require specific column types. For example, Ordinal Logistic requires a column with an Ordinal modeling type. If a column is added to the Y list that does not fit the personality that has been selected, JMP will change the personality to fit the data. The fitModelPersonalityConstants should be used to specify the personality type.

LaunchSpecifyRandomEffectMethod(method as fitModelRandomEffectMethods) As Boolean

Specify either REML (the recommended and default method) or EMS (the traditional method) approach. Returns True for success or False for failure.

UseByFit(Name As String) As Fit

Finds the By Group fit output associated with a given name, and returns the reference to that Fit object.

For example, suppose FitLeastSquares is launched on a group of people grouped by age. The Launch function returns a reference to the first FitLeastSquares object produced in the output. UseByFit(Name As String) As Fit can be used to return the references to the other output objects produced in the Launch. The type of object that is returned depends on the fitting personality that was originally selected for the analysis. For example, if the fitting personality was Ordinal, than a FitOrdinal object reference is returned by this method. Please note that this method is called from the original FitModel object, not the object that is returned from the Launch method call.

The Fit Model automation sample program has an example using this method.

FitNominal Object Methods

These methods provide access to options in the output for the Nominal fitting personality.

InversePrediction() As Boolean

Note that this action can't be turned off. InversePrediction() As Boolean brings up a dialog that requires user input. The values for InversePrediction() As Boolean cannot be supplied via automation.

LikelihoodRatioTests(Flag As Boolean) As Boolean

Turn the option on (True) or off (False).

OddsRatios(Flag As Boolean) As Boolean

Turn the option on (`True`) or off (`False`).

Profiler(Flag As Boolean)

Turns the Prediction Profiler on (`True`) or off (`False`).

ROCCurve(Flag As Boolean) As Boolean

Turn the option on (`True`) or off (`False`).

SaveProbFormula() As Boolean

Note that this action can't be turned off. The probability formula is saved to the current data table. `ConfidenceIntervals(Alpha As Double) As Boolean` supply the confidence intervals value.

FitOrdinal Object Methods

These methods provide access to options in the output for the Ordinal fitting personality.

ConfidenceIntervals(Double As Alpha) As Boolean

Supplies the confidence intervals value.

LikelihoodRatioTests(Flag As Boolean) As Boolean

Turns the option on (`True`) or off (`False`).

SaveExpectedValue() As Boolean

Saves the specified information in the current data table.

SaveProbFormula() As Boolean

Saves the specified information in the current data table.

SaveQuantiles() As Boolean

Saves the specified information in the current data table.

FitParametricSurvival Object Methods

These methods provide access to options in the output for the Parametric Survival fitting personality.

ConfidenceIntervals(Flag As Boolean) As Boolean

Turns the option on (`True`) or Off (`False`).

CorrelationOfEstimates(Flag As Boolean) As Boolean

Turns the option on (`True`) or Off (`False`).

CovarianceOfEstimates(Flag As Boolean)

Turns the option on (`True`) or off (`False`).

EstimateSurvivalProbability()

Brings up the interactive input fields for these options [EstimateSurvivalProbability() and EstimateTimeQuantile()]. Only one of these two options can be specified.

EstimateTimeQuantile()

Brings up the interactive input fields for these options [EstimateSurvivalProbability() and EstimateTimeQuantile()]. Only one of these two options can be specified.

LikelihoodRatioTests(Flag As Boolean) As Boolean

Turns the option on (`True`) or Off (`False`).

FitProfiler Object Methods

InteractionProfiler(Flag As Boolean)

Turns the option on (`True`) or off (`False`).

FitProportional Object Methods

The Proportional Hazards fitting model does not have methods that are unique to it. It does support the common analysis functions and therefore has its own object returned from the Fit Model Launch routine.

Methods

There are no methods specific to the FitProportional object.

FitResponse Object Methods

These provide access to Response-specific functions and tests. Examples are Effects Screening and Estimates.

BoxCoxY(Flag As Boolean) As Boolean

Turn the option on (`True`) or off (`False`). Information on this option can be found in the documentation of Fit Model in the *Fitting Linear Models* book.

CorrelationOfEstimates(Flag As Boolean) As Boolean

Turn the option on (`True`) or off (`False`). Information on this option can be found in the documentation of Fit Model in the *Fitting Linear Models* book.

ExpandedEstimates(Flag As Boolean) As Boolean

Turn the option on (`True`) or off (`False`). Information on this option can be found in the documentation of Fit Model in the *Fitting Linear Models* book.

GetEffectAnalysis(Name As BSTR) As FitEffect

Returns a reference to the various Effect analyses within the Response fitting when a `Standard Least Squares` analysis is launched. You can obtain a reference to each of these by calling this method and providing the name of the Effect that you wish to manipulate.

`FitEffect` object methods provide a way to manipulate Effect output returned using the object returned from `GetEffectAnalysis`. These correspond to the Effects popup menu in the analysis output.

You can also retrieve effects that contain crossed and nested terms. Examples might be `Silica*Silane*Sulfur` or `drug[Placebo,Gender]`.

InteractionPlots(Flag As Boolean) As Boolean

Turn the option on (`True`) or off (`False`). Information on this option can be found in the documentation of Fit Model in the *Fitting Linear Models* book.

LSMeansPlot(Flag As Boolean) As Boolean

Turn the option on (`True`) or off (`False`).

LSMeansStudents(Flag As Boolean) As Boolean

Turn the option on (`True`) or off (`False`).

LSMeansTable(Flag As Boolean) As Boolean

Turn the option on (`True`) or off (`False`).

LSMeansTukey(Flag As Boolean) As Boolean

Turn the option on (`True`) or off (`False`).

NormalPlot(Flag As Boolean) As Boolean

Turn the option on (`True`) or off (`False`). Information on this option can be found in the documentation of Fit Model in the *Fitting Linear Models* book.

ParameterPower(Flag As Boolean) As Boolean

Turn the option on (`True`) or off (`False`). Information on this option can be found in the documentation of Fit Model in the *Fitting Linear Models* book.

ParetoPlot(Flag As Boolean) As Boolean

Turn the option on (`True`) or off (`False`). Information on this option can be found in the documentation of Fit Model in the *Fitting Linear Models* book.

RowDiagnostics(`fitModelRowDiagConstants` diagType, VARIANT_BOOL Flag) As Boolean

Activates or deactivates the particular diagnostic. The first parameter is one of the available diagnostics taken from the `fitModelRowDiagConstants`. The `Flag` parameter turns the diagnostic on (`True`) or off (`False`).

SaveColumns(`fitModelSaveColumnConstants` **saveType) As Boolean**

Saves the selected output in a column, usually with the type of save as the prefix and the response name as the suffix of the new column. The type of save operations that can be used are contained in `fitModelSaveColumnConstants`.

ScaledEstimates(Flag As Boolean) As Boolean

Turn the option on (`True`) or off (`False`). Information on this option can be found in the documentation of Fit Model in the *Fitting Linear Models* book.

SequentialTests(Flag As Boolean) As Boolean

Turn the option on (`True`) or off (`False`). Information on this option can be found in the documentation of Fit Model in the *Fitting Linear Models* book.

TestSlices() As Boolean

Turns the Test Slices option on.

Notes:This is an action can't be turned off after it has been called.

FitStepwise Object Methods

These methods provide a way to drive Stepwise Regression in a similar way to the interactive approach.

AllPossibleModels()

Produces text display of all possible linear models using effects in the model.

AllPossibleModelsWithParameters(NMaximumTerms As Integer, NBestModelsToSee As Integer, HeredityRestriction As Boolean)

Copies the launch window for `AllPossibleModelsWithParameters`. `NMaximumTerms` is the maximum number of terms in the model. `NBestModelsToSee As Integer` is the number of best models to show. `HeredityRestriction As Boolean` enables or disables the `HeredityRestriction` condition.

EnterAll() As Boolean

Enters all unlocked effects into the model.

EnterEffect(EffectNumber As Integer, Flag As Boolean) As Boolean

Enters (`Flag = True`) or removes (`Flag = False`) the entry for the effect identified by the ordinal number provided in the first parameter.

GetEffectName(EffectNumber As Integer) As String

Returns a string with the name of the effect identified by the integer passed in as a parameter.

GetNumberOfEffects() As Short

Returns the number of effects in the Current Estimates table. This allows you to loop through the list of effects if you desire, and to obtain the names with `GetEffectName(EffectNumber As Integer) As String`.

Go() As Boolean

Starts the selection process. The process continues to run in the background until the model is finished.

LockEffect(EffectNumber As Integer, Flag As Boolean) As Boolean

Locks (`Flag = True`) or unlocks (`Flag = False`) the effect identified by the ordinal number provided in the first parameter.

RemoveAll() As Boolean

Removes (deselects) all effects from the model.

SetDirection(`fitStepDirectionConstants` Direction)

Allows the specification of how variables enter the regression equation. The direction constant should be one of the `fitStepDirectionConstants`. Possible values are `Forward`, `Backward`, or `Mixed`.

SetProbToEnter(Value As Double) As Boolean

Sets the Probability to Enter as a floating point value. See the documentation on Stepwise Regression in the *Fitting Linear Models* book for an explanation of these values.

SetProbToLeave(Value As Double) As Boolean

Sets the Probability to Leave as a floating point value. See the documentation on Stepwise Regression in the *Fitting Linear Models* book for an explanation of these values.

SetRules(`fitStepRulesConstants` Rules)

Allows the specification of the Rules value, just as in the stepwise dialog. The rules constant should be one of the `fitStepRulesConstants`. Possible values are `Combine`, `Restrict`, `No Rules`, and `Whole Effect`.

Step() As Boolean

Stops after each step of the stepwise process.

Stop() As Boolean

Stops the background selection process.

Gaussian Process Methods

The `Gaussian Process` object provides a way to launch and manipulate Gaussian process analyses.

ContourProfiler(Flag as Boolean)

Turn this option on (`True`) or off (`False`).

LaunchEstimateNuggetParameter(Flag as Boolean) As Boolean

Turn this launch option on (`True`) or off (`False`).

LaunchSpecifyCorrelationType(Type as jmpGaussianCorrelationConstant) As Boolean

Specify either Gaussian Process or Cubic Model for the correlation type.

LaunchSpecifyMinimumTheta(Theta as Double) As Boolean

Specify a Theta value prior to launch, just like in the Gaussian Process launch dialog.

Profiler(Flag as Boolean)

Turn this option on (`True`) or off (`False`).

SaveJacknifePredictedValues()

Save this formula to the current data table.

SavePredictionFormula()

Save this formula to the current data table.

SaveVarianceFormula()

Save this formula to the current data table.

SurfaceProfiler(Flag as Boolean)

Turn this option on (`True`) or off (`False`).

Hierarchical Cluster Methods

The Hierarchical Cluster platform has methods that apply only to Hierarchical Cluster. You can also use the Cluster Object Methods.

ColorMap(clusterColormapConstants mapType) As Boolean

Generates a color map of the values across the data range. This method mirrors the feature available from the Cluster platform.

DistanceGraph(Flag As Boolean) As Boolean

Turns the distance graph on (`True`) or off (`False`).

GeometricXScale(Flag As Boolean) As Boolean

Turns the Geometric X Scale option on (`True`) or off (`False`).

LaunchAddLabel(ColumnName As String) As Boolean

Adds a label column to the analysis. Returns `True` for success, `False` for failure.

LaunchAddOrdering(ColumnName As String) As Boolean

Adds an ordering column to the analysis. Returns `True` for success, `False` for failure.

LaunchRemoveLabel(ColumName As String) As Boolean

Removes a label column from the analysis. Returns `True` for success, `False` for failure.

LaunchRemoveOrdering(ColumnName As String) As Boolean

Removes an ordering column from the analysis. Returns `True` for success, `False` for failure.

SaveClusterHierarchy() As Boolean

Saves the information needed to do a custom dendrogram with scripting. For each cluster, this method returns three rows: the joiner, the leader, and the result, along with the cluster centers, size, and other information.

SaveDisplayOrder()

Saves the depth (order) of each row in a new data table column.

SetOrientation(clusterOrientationConstants orientation) As Boolean

Allows the specification of the dendrogram's orientation. It can be either `left`, `right`, `top` or `bottom`.

StandardizeData(Flag As Boolean)

A pre-launch option, this method should be called before the Launch method is invoked. It determines whether data is standardized by the column mean and standard deviation. The default is `True`, so call this with `False` if you don't want data standardized.

TwoWayClustering

Turns two way clustering on.

ItemAnalysis Object Methods

LaunchSpecifyModel(Model as itemAnalysisModelConstants)

Specifies the type of model to use, e.g. 2PL. This method must be used prior to calling the Launch method.

NumberOfPlotsAcross(Number as Integer)

Specifies the number of plots to be displayed horizontally.

SaveAbilityFormula()

Save the formula to the current data table.

KMeans Cluster Methods

The KMeans Cluster platform has methods that apply only to KMeans Cluster. You can also use the Cluster Object Methods.

KMGo()

Runs the cluster analysis.

Use `KMShiftDistances(Flag As Boolean)` and `KMWithinClusterStdDev(Flag As Boolean)` for *K*-means clustering *before* calling this method or `KMStep()`.

KMSaveMixtureFormulas()

Saves the mixture formulas in the current data table.

KMSaveMixtureProbs()

Saves the mixture probabilities in a new column of the active data table.

KMSeedWithSelectedRows() As Boolean

Specifies rows that contain values where you want the cluster centers to start. The rows in the data table must have been selected prior to invoking this method.

KMSimulateMixtures(long numberOfRows)

Mirrors the **Simulate Clusters** menu option. The `numberOfRows` parameter dictates how many rows are simulated.

KMShiftDistances(Flag As Boolean)

Specifies that points should give preference to being assigned to large clusters. The default is `False`.

Notes:Use this method and `KMWithinClusterStdDev(Flag As Boolean)` for *K*-means clustering *before* calling `KMGo()` or `KMStep()`.

KMStep()

Performs one iteration of the clustering, to allow inspection of the values.

Use `KMShiftDistances(Flag As Boolean)` and `KMWithinClusterStdDev(Flag As Boolean)` for *K*-means clustering *before* calling this method or `KMGo()`.

KMWithinClusterStdDev(Flag As Boolean)

This standardizes the distance components by the within-cluster standard deviation. The default is `False`.

Use this method and `KMShiftDistances(Flag As Boolean)` for *K*-means clustering *before* calling `KMGo()` or `KMStep()`.

LaunchAddFreq(ColumnName As String) As Boolean

Adds a frequency column to the analysis. Returns `True` for success, `False` for failure.

LaunchAddWeight(ColumnName As String) As Boolean

Adds a weight column to the analysis. Returns `True` for success, `False` for failure.

LaunchRemoveFreq(ColumName As String) As Boolean

Removes an frequency column from the analysis. Returns `True` for success, `False` for failure.

LaunchRemoveWeight(ColumnName As String) As Boolean

Removes a weight column from the analysis. Returns `True` for success, `False` for failure.

StandardizeData (Flag As Boolean)

Specifies to standardize (`True`) or not standardize (`False`) the data. This method can be used before or after the call to launch the platform.

Logistic Object Methods

The `Logistic` object provides a way to launch and manipulate a logistic regression analysis.

InversePrediction()

Request an inverse prediction and produces the Inverse Prediction Dialog Box.

LiftCurve(Flag as Boolean)

Display the lift curve or lift chart (`True`) or turns the display off (`False`).

LineColor(Color as jmpColorConstants)

Changes the color of the logistic lines.

LogisticPlot(Flag As Boolean)

Turns the logistic plot on (`True`) or off (`False`).

NomAxisBooleanOption(Handle as Long, Action as Short, Flag As Boolean)

Controls a Boolean option for the display of the Nominal Axis. If the platform supports the option, then this method will either turn it on (`Flag` is `True`) or off (`Flag` is `False`). Examples of options are Rotated Tick Labels, Divider Bars and displaying a Lower Frame. The Rotated Tick Labels are supported only on the Oneway and Variability Chart platforms. Before this method can be called, a handle to the Nominal Axis display box must be obtained through a call to `GetGraphicItemByType(TypeName As String, ItemNumber As Integer) As Long`.

RateCurve()

Turns the Rate Curve on.

ROCCurve(Flag As Boolean)

Creates a ROC curve.

ROCSetPositiveLevel(LevelValue As String) As Boolean

Sets the positive level value (a value within the Y variable) prior to calling `ROCCurve`. Returns `True` on success, `False` on failure. If this method is called after `ROCCurve` is called, nothing happens.

MatchedPairs Object Methods

The `MatchedPairs` object provides a way to launch a matched pairs analysis. It also supports the common analysis methods.

SignTest(Flag As Boolean)

Turns the Sign Test on (`True`) or off (`False`).

SetAlphaLevel(Alpha As Double)

Sets the alpha value.

WilcoxonSignedRank(Flag As Boolean)

Turns Wilcoxon Signed Rank on (True) or off (False).

Measurement Systems Analysis (MSA)

The `MeasurementSystemsAnalysis` object provides a way to launch an analysis of a measurement system. Beyond the typical Launch methods for specifying variables, the following methods are supported:

BiasStudySetAlpha(alpha as Double) As Boolean

Sets the alpha value for the Bias study report. This method is available only after the Launch platform is called.

BiasStudySetChartOptions(option as MSAStudyChartOptions) As Boolean

Set the display options for the Bias study chart output. Examples include limits shading and needle plots.

LaunchSpecifyAnalysisSettings(maxIterations as Integer, convergenceLimit as Double) As Boolean

Mimics the options available in the MSA launch dialog.

LaunchSpecifyAlpha(Alpha as Double) As Boolean

Specifies a pre-launch Alpha value.

LaunchSpecifyChartDispersionOptions(option as MSAChartDispersionTypes) As Boolean

Allows for the specification of Range or Standard Deviation type for the chart dispersion choice.

LaunchSpecifyModelOptions(option as MSAModelTypes) As Boolean

Allows `Nested` or `Crossed` to be specified for the Model option. Turns the option on (`True`) or Off (`False`), as when selected in the **Measurement Systems Analysis** menu.

The following methods for manipulating the output, either before or after launch, are available:

- **ShowAverageChart**
- **ShowRangeChart**
- **ParallelismStudy**
- **EMPStudy**
- **MeasurementIncrementStudy**
- **VarianceComponents**
- **EMPGaugeStudy**
- **ShowBiasStudy**
- **ShowTestRetestErrorStudy(Flag as Boolean)**

RangeChartOption(option as MSARangeChartOptions) As Boolean

Turns on (True) or off (False) options associated with the Range chart. An example is "Show Average Range". If the Range chart is not the current type of dispersion chart, an error is shown in the log.

StandardDeviationChartOption(option as MSAStandardDeviationChartOptions) As Boolean

Turns on (`True`) or off (`False`) options associated with the Standard Deviation chart. An example is "Show Average Range". If the StandardDeviation chart is not the current type of dispersion chart, an error is shown in the log.

TestRetestStudySetChartOptions(option as MSAStudyChartOptions) As Boolean

Sets the display options for the Test/Re-test study chart output. Examples include limits shading and needle plots.

Methods for Manipulating the Output

The following methods manipulate the output, either before or after launch:

ShiftDetectionProfiler(Flag as Boolean)

Turns this option on (True) or of (False).

AverageChartOption(option as MSAStandardDeviationChartOptions) As Boolean

Turns Average chart options on (True) or off (False). `Show Limits` is an example of an Average chart option.

ShowStandardDeviationChart(Flag as Boolean)

Turns the Deviation chart on (True) or off (False).

Multiple Correspondence Analysis

Before Launch()

The following methods need to be called before the Launch() method:

LaunchAddResponse(name As String) As Boolean

Assigns a column as a response variable. Name refers to a data column. Returns `True` upon success and `False` upon failure.

Launch() As Boolean

Launches the MCA report. Returns `True` upon success and `False` upon failure.

LaunchAddFactor(name As String) As Boolean

Assigns a column as a factor. `Name` refers to a data column. Returns `True` upon success and `False` upon failure.

LaunchAddSupplementaryVariable(name As String) As Boolean

Assigns a column as a supplementary variable. `Name` refers to a data column. Returns `True` upon success and `False` upon failure.

LaunchAddSupplementaryID(name As String) As Boolean

Assigns a column as a supplementary ID. `Name` refers to a data column. Returns `True` upon success and `False` upon failure.

LaunchAddFreq(name As String) As Boolean

Assigns a column as a frequency. `Name` refers to a data column. Returns `True` upon success and `False` upon failure.

LaunchAddBy(name As String) As Boolean

Assigns a column as a by variable. `Name` refers to a data column. Returns `True` upon success and `False` upon failure.

LaunchRemoveResponse(name As String) As Boolean

Removes a column as a response. `Name` refers to a data column. Returns `True` upon success and `False` upon failure.

LaunchRemoveFactor(name As String) As Boolean

Removes a column as a factor. `Name` refers to a data column. Returns `True` upon success and `False` upon failure.

LaunchRemoveSuplementaryVariable(name As String) As Boolean

Removes a column as a supplementary variable. `Name` refers to a data column. Returns `True` upon success and `False` upon failure.

LaunchRemoveSupplementaryID(name As String) As Boolean

Removes a column as a Supplementary ID. `Name` refers to a data column. Returns `True` upon success and `False` upon failure.

LaunchRemoveFreq(name As String) As Boolean

Removes a column as a frequency. `Name` refers to a data column. Returns `True` upon success and `False` upon failure.

LaunchRemoveBy(name As String) As Boolean

Removes a column as a by variable. `Name` refers to a data column. Returns `True` upon success and `False` upon failure.

After Launch()

The following methods need to be called after the Launch() method:

CrossTable(flag As Boolean)

Provides the Burt table or contingency table as appropriate for variable roles selected. `Flag` turns the option on (`True`) or off (`False`).

DisplayOptions(option As MCADisplayOptions, flag as Boolean)

Turns the various display options on or off for the MCA report. Option refers to the display option to toggle. `Flag` turns the option on (`True`) or off (`False`).

SaveCoordinates(nDims As Short)

Saves the principal coordinates to one or more JMP data tables. `nDims` refers to the number of columns to save.

SaveCoordinateFormula(nDims As Short)

Saves formula columns to the source data table for the principal coordinates in multiple dimensions. `nDims` refers to the number of columns to save.

Multivariate Object Methods

The `Multivariate` object provides a way to launch and manipulate a multivariate analysis and the automation objects that it creates.

ColorMapOnCorrelations(Flag As Boolean)

Show a color map based on correlations (`True`) or hide it (`False`).

ColorMapOnValues(Flag As Boolean)

Show a color map based on *p*-values (`True`) or hide it (`False`).

ClusterOnCorrelations(Flag As Boolean)

Group variables that have similar correlations in a color map based on correlations (`True`) or do not group (`False`).

CorrelationProbability, CIofCorrelation(Flag As Boolean)

Turn the option on (`True`) or off (`False`).

CorrelationsM(Flag As Boolean)

Display option that can be set by specifying `True` for parameter, or reset by specifying `False`. This defers to the `CorrelationsMultivariate` option.

CovarianceMatrix(Flag As Boolean)

Display option that can be set by specifying `True` for parameter, or reset by specifying `False`.

CronbachsAlpha(Flag As Boolean)

Display option that can be set by specifying `True` for parameter, or reset by specifying `False`.

Ellipsoid3D(BSTR X, BSTR Y, BSTR Z) As Boolean

Create a 3D ellipsoid given the 3 columns. If the function returns `False`, it indicates failure. This most likely is the result of an invalid column name. `True` indicates success.

HoeffdingsD(Flag As Boolean)

Display option that can be set by specifying `True` for parameter, or reset by specifying `False`.

InverseCorr(Flag As Boolean)

Display option that can be set by specifying `True` for parameter, or reset by specifying `False`.

KendallsTau(Flag As Boolean)

Display option that can be set by specifying `True` for parameter, or reset by specifying `False`.

MultivariateSimpleStatistics(Flag as Boolean)

Displays the Multivariate Simple Statistics report (`True`) or turns the display off (`False`).

OutlierAnalysis(Flag As Boolean) As Outlier

Creates an outlier analysis, and returns a dispatch pointer to a `Outlier` object that can be manipulated further. (See “Outlier Object Methods” on page 153.)

PairwiseCorr(Flag As Boolean)

Displays option that can be set by specifying `True` for parameter, or reset by specifying `False`.

ParallelCoordPlot(Flag as Boolean)

Displays the Parallel Coordinate Plot (`True`) or turns the display off (`False`).

PartialCorr(Flag As Boolean)

Display option that can be set by specifying `True` for parameter, or reset by specifying `False`.

PrincipalOnCorrelations As PrincipalComponents

Performs a principal components analysis and returns a dispatch pointer to a `PrincipalComponents` object that can be manipulated further. (See "PrincipalComponents Object Methods" on page 159.)

PrincipalOnCovariances As PrincipalComponents

Performs a principal components analysis and returns a dispatch pointer to a `PrincipalComponents` object that can be manipulated further.

PrincipalUncentered() As PrincipalComponents

Performs a principal components analysis and returns a dispatch pointer to a `PrincipalComponents` object that can be manipulated further.

SaveTSquare()

Save the TSquare distances to the current data table.

ScatterPlot(Flag As Boolean) As ScatterPlotMatrix

Creates a scatterplot matrix and returns a dispatch pointer to the `ScatterPlotMatrix` object to allow further manipulation. (See "ScatterPlotMatrix Object Methods" on page 161.)

SpearmansRho(Flag As Boolean)

Display option that can be set by specifying `True` for parameter, or reset by specifying `False`.

StandardizedAlpha(Flag As Boolean)

Display option that can be set by specifying `True` for parameter, or reset by specifying `False`.

TSquareDistances(Flag As Boolean)

Displays the TSquare distances (`True`) or turn the display off (`False`).

UnivariateSimpleStatistics(Flag as Boolean)

Display the Univariate Simple Statistics report (`True`) or turns the display off (`False`).

Multivariate Control Chart Object Methods

Automation of the Multivariate Control Chart platform has been added. The methods available on the platform mirror those that are available in the analysis output window for Multivariate Control Chart.

PrincipalComponents(Flag as Boolean)

Turn the post-launch option for principal components on (`True`) or off (`False`).

SavePrincipalComponents()

Saves the data for principal components in a new column in the current data table.

SaveTargetStatistics()

Saves the data for target statistics in a new column in the current data table.

SaveTSquare()

Saves the data for T square in a new column in the current data table.

ShowCorrelation(Flag as Boolean)

Turn the post-launch option to show the correlation table on (`True`) or off (`False`).

ShowCovariance(Flag as Boolean)

Turn the post-launch option to show the covariance on (`True`) or off (`False`).

ShowInverseCorrelation(Flag as Boolean)

Turn the post-launch option to show the inverse correlation table on (`True`) or off (`False`).

ShowInverseCovariance(Flag as Boolean)

Turn the post-launch option to show the inverse covariance on (`True`) or off (`False`).

ShowMeans(Flag as Boolean)

Turn the post-launch option to show the means on (`True`) or off (`False`).

Neural Object Methods

The Neural object methods provide a way to launch and manipulate the Neural analysis platform.

ControlPanelOptions(neuralControlConstants option, Flag As Boolean)

This allows any of the Neural control panel check box options to be selected or deselected. The first parameter is a constant that specifies the desired checkbox option, and the second parameter specifies whether the option should be turned On (`True`) or Off (`False`). These options are then used in the Neural calculations when the `Go()` method is invoked. By default, all the options are Off.

Diagram(Flag As Boolean)

Turns the Neural diagram On (`True`) or Off (`False`). It is Off by default.

Go()

Starts the Neural calculations.

Profiler(Flag As Boolean)

Turns the prediction profiler On (`True`) or Off (`False`). It is Off by default.

SaveHidden()

Saves the specified data in the current data table. Mimics the **Save Hidden and Scaled Cols** menu option.

SaveFormulas()

Saves the specified data in the current data table. Mimics the **Save Formulas** menu option.

SavePredicted()

Saves the specified data in the current data table. Mimics the **Save Predicted** menu option.

SaveProfileFormulas()

Saves the specified data in the current data table. Mimics the **Save Profile Formulas** menu option.

SpecifyConvergeCriterion(Value As Double) As Boolean

This mimics the control panel option in the regular UI. The value is used in the Neural calculations when the `Go()` method is invoked.

SpecifyHiddenNodes (Value As Double) As Boolean

This mimics the control panel option in the regular UI. The value is used in the Neural calculations when the `Go()` method is invoked.

SpecifyMaxIterations(Value As Double) As Boolean

This mimics the control panel option in the regular UI. The value is used in the Neural calculations when the `Go()` method is invoked.

SpecifyNumberOfTours(Value As Double) As Boolean

This mimics the control panel option in the regular UI. The value is used in the Neural calculations when the `Go()` method is invoked.

SpecifyOverfitPenalty(Value As Double) As Boolean

This mimics the control panel option in the regular UI. The value is used in the Neural calculations when the `Go()` method is invoked.

Normal Mixtures Methods

The Normal Mixtures automation platform replaces the old KMNormalMixtures method in the Cluster platform.

Biplot, Biplot3D, ParallelCoordinatePlot, ScatterPlotMatrix(Flag as Boolean)

Turn these options on (`Flag` is `True`) or off (`Flag` is `False`) after calling the `Go()` method.

BiplotContourDensity(Density as Double)

Specifies the Biplot contour density with a value between 0 and 1.

Go()

Runs the Normal Mixtures with all of the control panel settings that have been specified using any values set with the **Specify** methods, and with default values that were not changed with a **Specify** call.

LaunchAddY, LaunchAddFreq, LaunchAddWeight, LaunchAddBy

These functions, as in other automation platforms, add variables to the respective roles prior to calling `Launch()`.

PublishClusterFormulas()

Saves the cluster formulas to the Formula Depot, where they can be exported to a variety of other languages such as Python or C.

SaveClusterFormula, SaveMixtureFormulas, SaveMixtureProbabilities, SaveDensityFormula()

Save columns in the original data table with the appropriate formula values.

SaveColorsToTable

Saves the cluster colors to the data table as row state colors.

SimulateClusters(NRows as Integer)

Creates simulated data. Specify how many observations you would like.

SpecifyNClusters(NClusters as Integer), SpecifyNTours(NTours as Integer), SpecifyMaxiumumIterations(MaxIterations as Integer), SpecifyConvergeCriterion(Criterion as Double)

Use these methods prior to calling `Go()` on the Normal Mixtures object. The methods mirror the options that you see in the control panel for Normal Mixtures immediately after launching the platform.

Oneway Object Methods

The `Oneway` object provides a way to launch and manipulate a Oneway (Fit Y by X) analysis.

AnalysisOfMeans(Type as OnewayAofMConstants, Flag As Boolean)

Performs an Analysis of Means given the ANOM type. Some Analysis of Means types have specific constraints regarding their usage. If the analysis fails, view the log for an explanation.

CDFPlot(Flag As Boolean)

Displays a CDF Plot (`True`) or hides it (`False`).

CompareDensities(Flag As Boolean)

Displays the Compare Densities outline (`True`) or hides it (`False`).

CompareMeans(Option As Integer, Flag As Boolean)

Turns on or off the means comparison operation specified in the first parameter. This should be a value from the `OnewayCompareConstants` definition. The flag indicates on (`True`) or off (`False`).

CompositionOfDensities(Flag As Boolean)

Displays the Composition of Densities outline (`True`) or hides it (`False`).

DisplayOptions(Option As Integer, Flag As Boolean)

Turns on (*`Flag`* = `True`) or off (*`Flag`* = `False`) a variety of display options for the analysis graphics. The `Option` parameter should be a value from the `OnewayDisplayConstants` definition.

EquivalenceTest(diffConsideredPracticallyZero as Double)

Performs an equivalence test using the value that is to be treated as the difference. This difference is considered to be practically zero.

FitRobust, FitCauchy(Flag As Boolean)

Turns the Robust or Cauchy Fit on (`True`) or off (`False`).

Histograms(Flag as Boolean)

Displays the histograms for each column in the analysis next to the oneway graph.

Kolmogorov Smirnov

Displays the Kymograph Smyrna nonparametric test.

MatchingColumn(ColumnName As String) As Boolean

Allows you to do a matching model analysis with the variable (column) provided. Returns `True` if successful, `False` if the column doesn't exist or there is some other error.

MeansAnovaT(Flag As Boolean)

Provides a way to show (`True`) or hide (`False`) this additional analysis output.

MeansStdDev(Flag As Boolean)

Provides a way to show (`True`) or hide (`False`) this additional analysis output.

NomAxisBooleanOption(Handle as Long, Action as Short, Flag As Boolean)

Controls a Boolean option for the display of the Nominal Axis. If the platform supports the option, then this method will either turn it on (`Flag` is `True`) or off (`Flag` is `False`). Examples of options are Rotated Tick Labels, Divider Bars and displaying a Lower Frame. The Rotated Tick Labels are supported only on the Oneway and Variability Chart platforms. Before this method can be called, a handle to the Nominal Axis display box must be obtained through a call to `GetGraphicItemByType(TypeName As String, ItemNumber As Integer) As Long`.

Nonparametric(Option As Integer, Flag As Boolean)

Produces or hides nonparametric tests. The test type is determined by the first parameter, which should be a value from the `OnewayNonParConstants` definition. The second parameter indicates whether to show (`True`) or hide (`False`) the display output.

NonParametricMultipleComparisons(Type as OnewayNonParMultipleComparisonConstants, Flag As Boolean)

Performs the comparison specified by `Type if Flag` is `True`, otherwise turn off the comparison. A dialog appears for comparisons that require a control value If you do not want the dialog to appear, use the `NonParametricMultipleWithControl` method.

NonParametricMultipleWithControl(Type as OnewayNonParMultipleComparisonConstants, ControlValue as String) As Boolean

Performs the comparison where a control value needs to be specified. Examples are “Steel with Control” and “Dunn with Control for Joint Ranks”. The control value can be a numeric or character value depending on the column type of the X value, so the control must be specified as a string even if the value is a number. Examples are “Female” or “15”.

NormalQuantileLineOfFit(Flag As Boolean)

Allows you to turn on or off the `Line of Fit` option found on the **Normal Quantile Plot** submenu.

NormalQuantilePlot(Flag As Boolean)

Allows you to turn on or off the `Plot Actual By Quantile` option found on the **Normal Quantile Plot** submenu.

NormalQuantilePlotQbyA(Flag As Boolean)

Allows you to turn on or off the `Quantile by Actual` option found on the **Normal Quantile Plot** submenu.

NormalQuantileProbLabels(Flag As Boolean)

Allows you to turn on or off the `Probability Labels` option found on the **Normal Quantile Plot** submenu.

ProportionOfDensities(Flag as Boolean)

Turns the option on (`True`) or off (`False`).

Save(Option As Integer)

Allows you to save standard, centered or normalized quantiles to a new column in the data table. The Option parameter specifies the type of save and should be a value from the `OnewaySaveConstants` definition. See the *Basic Analysis* book for further details.

SetAlpha(Level As Double)

Specify the alpha level, e.g. 0.95.

TTest(Flag as Boolean)

Turns on/off the t-test analysis when comparing two groups.

UnequalVariances(Flag As Boolean)

Provides a way to show (`True`) or hide (`False`) this additional analysis output.

Outlier Object Methods

The `Outlier` object is produced from the `OutlierAnalysis(Flag As Boolean) As Outlier` method of the `Multivariate` object.

JacknifeDistances(Flag As Boolean)

Turns the display options on (`True`) or off (`False`).

MahalanobisDistances(Flag As Boolean)

Turns the display options on (`True`) or off (`False`).

SaveJacknife()

Saves the distances into a new column in the data table.

SaveMahal()

Saves the distances into a new column in the data table.

Overlay Object Methods

The `Overlay` object provides a way to launch and manipulate overlay charts. The original Overlay platform in JMP has been deprecated. Automation code that calls the Overlay platform will now use the equivalent Graph Builder commands to generate output. There may be differences in appearance from prior versions.

LaunchAddYWithRightScale(ColumnName As String)

Adds a Y variable, with Right Scaling turned on. If you want Left Scaling, just use `LaunchAddY`.

LaunchSetSortScaleOptions(XSort as Boolean, XLogAxis as Boolean, YLogLeftAxis As Boolean, YLogRightAxis As Boolean)

Turns the X axis sort on or off, turns the X Axis Log scaling on or off, and turns the left and right Y Axis Log scaling on or off (`True` or `False`). This mirrors the options in the **Overlay Plot** launch dialog.

LineOptions(overlayLineStyleConstants style, overlayLineThicknessConstants thickness)

Sets the line type and thickness when the ConnectPoints option is specified.

Overlay(Flag As Boolean)

Specifies if you want an Overlay plot (`True`) or individual plots (`False`).

Range(Flag As Boolean)

Specifies if you want a range plot (`True`) or not (`False`).

SeparateAxes

Display option for the overlay plot that can be turned on (`True`) or off (`False`).

YConnectColor(Color As jmpColorConstants)

Sets the color of the connecting line between points.

YConnectPoints(Flag As Boolean)

Display option for the overlay plot that can be turned on (`True`) or off (`False`).

YOverlayMarker(Marker as jmpMarkerConstants)

Sets the type of marker used for points in the Overlay Plot.

YOverlayMarkerColor(Color as jmpColorConstants)

Sets the marker color for points in the Overlay Plot.

YNeedle(Flag As Boolean)

Display option for the overlay plot that can be turned on (`True`) or off (`False`).

YShowPoints(Flag As Boolean)

Display option for the overlay plot that can be turned on (`True`) or off (`False`).

YStep(Flag As Boolean)

Display option for the overlay plot that can be turned on (`True`) or off (`False`).

Parallel Plot Methods

The `Parallel` object provides a way to launch and manipulate parallel charts.

ReverseScaleOnY(ColumnName as String) As Boolean

Reverse the scale on one of the Y columns that was specified prior to Launch. Returns `True` if successful, `false` if it fails to find the column.

ShowReverseCheckboxes(Flag as Boolean)

Display the checkboxes for reversing the scaling on Y values.

Pareto Object Methods

The `Pareto` object provides a way to launch and manipulate pareto charts.

AddCauseToCombine(causeName As String)

Adds a cause name to a list that is used to accumulate all the causes that should be combined. Once all the causes have been added with this method, call `CombineCauses() As Boolean` to combine them all.

CategoryLegend(Flag As Boolean)

This is a display option corresponding to a Pareto menu option. It can be turned on (`True`) or off (`False`).

CombineCauses() As Boolean

Combine all the causes added with `AddCauseToCombine(causeName As String)`. Returns `True` for success, `False` for failure.

CumPercentAxis(Flag As Boolean)

This is a display option corresponding to a Pareto menu option. It can be turned on (`True`) or off (`False`).

CumPercentCurve(Flag As Boolean)

This is a display option corresponding to a Pareto menu option. It can be turned on (`True`) or off (`False`).

CumPercentPoints(Flag As Boolean)

This is a display option corresponding to a Pareto menu option. It can be turned on (`True`) or off (`False`).

HorizontalLayout(Flag As Boolean)

This is a display option corresponding to a Pareto menu option. It can be turned on (`True`) or off (`False`).

Nlegend(Flag As Boolean)

This is a display option corresponding to a Pareto menu option. It can be turned on (`True`) or off (`False`).

PercentScale(Flag As Boolean)

This is a display option corresponding to a Pareto menu option. It can be turned on (`True`) or off (`False`).

PieChart(Flag As Boolean)

This is a display option corresponding to a Pareto menu option. It can be turned on (`True`) or off (`False`).

SeparateCauses()

Separate all the causes that are currently combined.

UngroupPlots(Flag As Boolean)

Turns the option on (`True`) or off (`False`).

Partition Object Methods

Automation of the Partition platform has been added. The methods available on the platform mirror those that are available in the analysis output window for Partition.

ColorPoints(Flag as Boolean)

Turn the option to color points on (`True`) or off (`False`). This method will only work when the Y variable is Nominal or Ordinal.

ColumnContributions(Flag as Boolean)

Turn the option to show Column Contributions on (`True`), or off (`False`).

Criterion(Option as partitionCriterionConstants, Flag as Boolean)

Select one of the criteria from a predefined list of constants.

DisplayOptions(Option as partitionDisplayConstant, Flag as Boolean)

Select a display option from one of the predefined constants and turn it on (`True`) or off (`False`).

KFoldCrossValidation(value as Integer)

Specify the K value as an integer.

LeafReport(Flag as Boolean)

Turn the option to show the Leaf Report on (`True`), or off (`False`).

LiftCurve(Flag as Boolean)

Turn the option to show the Lift Curve on (`True`) or off (`False`). This method will only work when the Y variable is Nominal or Ordinal.

LockColumns(Flag as Boolean)

Turn the option to lock the columns on (`True`), or off (`False`).

MinimizeSizeSplit(value as double)

Specify the minimum value as a double.

MissingValueRule(Option as partitionMissingConstants, Flag as Boolean)

Select one of the rules for treating missing values from the predefined constants.

PlotActualByPredicted(Flag as Boolean)

Turn the option to show Plot Actual by Predicted on (`True`), or off (`False`).

Prune()

Performs the Prune Worst function on the Partition.

ROCCurve(Flag as Boolean)

Turn the option to show the ROC Curve on (`True`) or off (`False`). This method will only work when the Y variable is Nominal or Ordinal.

SaveColumns(SaveOperation as partitionSaveColumnConstants)

Save a column of information in the current data table. The information that is saved is determined by the value that is passed into parameter 1 from the predefined constants.

SmallTreeView(Flag as Boolean)

Turn the option to show the Small Tree View on (`True`), or off (`False`).

Split()

Performs the Split Best function on the partition.

SplitHistory(Flag as Boolean)

Turn the option to show the Split History on (`True`), or off (`False`).

Partial Least Squares Object Methods

The PLS object provides a way to launch and manipulate Partial Least Squares analyses. The original PLS platform that this automation platform was designed around is being removed from the product. There is a new, much more fully featured, Partial Least Squares platform in JMP. The automation support will attempt to map the existing automation API to use the new JMP platform where possible. Users of the previous PLS automation should be careful to examine the new output to make sure it meets their needs.

"Before Launch()" on page 157 and "After Launch()" on page 158 describe methods for the Partial Least Squares platform. "Legacy PLS methods" on page 158 describes the older PLS methods.

Before Launch()

The following methods need to be called before the Launch() method:

LaunchSpecifyModelMethod(Method As plsModelMethodConstants)

Defines the model type using one of predefined constants. Currently, NIPALS and SIMPLS are supported.

LaunchSpecifyValidationType(valType As plsValidationTypes, valParm As Double)

Specifies the validation type (for example, KFold or Holdback). The second parameter is used for methods, such as KFold, that take a value for Number of Folds. The second parameter is also used for Holdback, which takes a value for Holdback Proportion. See the PLS chapter in the *Multivariate Methods* book for more information.

LaunchSpecifyInitialNumberOfFactors(nFactors As Int)

Specifies the number of factors before starting the fit, just as in the window for PLS. Specify the number of factors after entering the factors using `LaunchAddX`.

LaunchSetRandomSeed(Seed As Double)

Sets an optional random seed. The default method does not use a seed.

LaunchAddValidationColumn(Name As String)

You can specify one validation column.

LaunchRemoveValidationColumn(Name As String)

Removes the validation column prior to calling Launch.

LaunchSpecifyOptions(Centering As Boolean, Scaling As Boolean)

Turns the Centering or Scaling options on (True) or off (False). By default, the options are on.

LaunchSpecifyImputeMethod(Method As plsImputMethods, Iterations As Int)

Specifies a method for missing value imputation using a predefined constant. For the EM method, you can specify the maximum number of iterations in the second parameter. The second parameter, while required, is ignored for the Mean method.

After Launch()

The following methods can be called after the Launch() method has been called:

PercentVariationPlots, LoadingScatterPlotMatrices, Profiler, VIPVersusCoefficientPlots, CoefficientPlots, ScoreScatterplotMatrices, SpectralProfiler(Flag As Boolean)

Turns the option on (`True`) or off (`False`).

CorrelationLoadingPlot(Int nFactors)

Shows a Correlation Loading Plot, specifying the number of factors.

Legacy PLS methods

ConfidenceLines(Flag As Boolean)

Turn the option on (`True`) or off (`False`).

SaveFormula()

Saves the prediction formula to the current data table.

SaveOutputs(Flag As Boolean)

Turn the option on (`True`) or off (`False`).

ShowPoints(Flag As Boolean)

Turn the option on (`True`) or off (`False`).

PrincipalComponents Object Methods

The PrincipalComponents object is produced by `PrincipalOnCorrelations As PrincipalComponents`, `PrincipalOnCovariances As PrincipalComponents`, `PrincipalUncentered() As PrincipalComponents` methods of the Multivariate object.

FactorRotation(N As Integer)

Performs a factor rotation with `N` factors.

SavePrincipal(Num As Integer)

Saves `Num` components as data table columns.

SaveRotated()

Saves rotated factors in a new column of the data table.

Spin(Flag As Boolean)

Invokes the spin plot of the first three principal components if `Flag` is `True`.

Profiler Object Methods

The Profiler object provides a way to launch and manipulate a prediction profiler chart.

ConfidenceIntervals(Flag As Boolean)

Turns on (`True`) or off (`False`) the Confidence Interval display option.

Desirability(Flag As Boolean)

Turns on (`True`) or off (`False`) the Desirability Functions display option.

InteractionProfiler(Flag as Boolean)

Turns the option on (`True`) or off (`False`).

LaunchAddNoiseFactors(NoiseFactorsColumn As String) As Boolean

Add a column for noise factors to study robustness.

MostDesirable()

Executes the **Most Desirable in Grid** operation.

Recurrence Object Methods

The `Recurrence` object provides a way to launch and manipulate a Recurrence analysis.

EventPlot(Flag As Boolean)

Determines whether the Event plots are shown (`True`) or hidden (`False`).

MCFConfidLimits(Flag As Boolean)

Controls the MCF confidence limits display option, either showing the limits (`True`) or hiding them (`False`).

MCFPlot(Flag As Boolean)

Determines whether the MCF plots are shown (`True`) or hidden (`False`).

PlotMCFDifferences(Flag as Boolean)

Turns the option on (`True`) or off (`False`).

Scatterplot3D Object Methods

Scatterplot 3D automation supports most of the features available through the menus.

BiplotRays(Flag As Boolean)

Show biplot rays (`True`) or hide biplot rays (`False`). Biplot rays will only be visible if an option that normally produces biplot rays has been run.

ConnectPoints(BSTR groupingColumn)

Connect the points in the plot. If you do not wish to supply a grouping column, you must specify an empty string ("").

DropLines(Flag As Boolean)

Show drop lines (`True`) or hide drop lines (`False`).

NormalContourEllipsoids(BSTR groupingColumn)

Show normal contour ellipsoids. If you do not wish to supply a grouping column, you must specify an empty string ("").

PrincipalComponents()

Turn principal components on.

RotatedComponents()

Open a dialog with a variety of parameters for specifying factoring and rotation methods.

SavePrincipalComponents(Number as Long)

Save a number of principal components, specified by the input parameter, to the current data table.

SaveRotatedComponents()

If `RotatedComponents()` has already been run, this saves the component values to the current data table.

ShowPoints(Flag As Boolean)

Show points (`True`) or hide points (`False`).

StdPrincipalComponents()

Turn standard principal components on.

ScatterPlotMatrix Object Methods

The `ScatterPlotMatrix` object is produced by the `ScatterPlot(Flag As Boolean) As ScatterPlotMatrix` method of the `Multivariate` object.

DensityEllipses(Flag As Boolean)

Turns this display option on (`True`) and off (`False`).

EllipseAlpha(Alpha As Double)

Specifies the percentage of points that should be enclosed in the ellipse if it is normally distributed.

EllipseColor(Color As Integer)

Specifies the ellipse color, from one of the `jmpColorConstants` values.

Histograms(HorizontalHistogram As Boolean, Flag As Boolean)

Displays a Histogram in the scatterplot matrix. If the first parameter is `True`, a Horizontal Histogram is display, if `False` a Vertical one is displayed. The flag turns the Histogram on (`True`) or off (`False`).

Scatterplot Matrix Object Methods

These methods support the Scatterplot Matrix platform, not the `ScatterPlot(Flag As Boolean) As ScatterPlotMatrix` method of the `Multivariate` object.

DensityEllipses(Flag As Boolean)

Turn the ellipses on (`True`) or off (`False`).

EllipseAlpha(Alpha as Double)

Specify the ellipse alpha value, between 0.0 and 1.0.

EllipseTransparency(Transparency As Double)

Sets the ellipse transparency value. The value must be between 0 and 1.

LaunchSpecifyMatrixFormat(scatterplotMatrixFormatconstants val)

Specify the format of the scatterplot matrix (Lower Triangular, Square etc.) prior to calling the Launch method.

ShowCorrelations, ShowPoints, FitLine, NonParDensity (Flag As Boolean)

Turns the option on (`True`) or off (`False`).

Screening Object

The Screening object provides a way to launch the Screening platform.

Methods

There are no methods specific to the Screening object.

SpinPlot Object Methods

The `SpinPlot` object provides a way to launch and manipulate a spinning plot. After the spin plot is created through a Launch, the `Spin` method must be called to animate the plot.

Notes:SpinPlot has been removed as of JMP 8. Code that was written to use SpinPlot will continue to work, but will invoke the Scatterplot 3D platform instead. All existing methods on the SpinPlot automation interface should continue to work.

BiplotRays(Flag As Boolean)

Toggles the biplot ray display option on (`True`) or off (`False`).

PrincipalComponents()

This option mirrors its non-automation counterpart, calculating principal components on the launch variables.

RotatedComponents(Number As Integer)

Computes *`Number`* rotated component scores.

SavePrincipalComponents()

Saves the current principal component in a new column of the data table.
`SaveToPrincipalComponents` will produce a prompt asking for the number of principal components to save. You can use `SavePrincipalComponents2(NumberToSave as Short)` to specify the number of principal components to save, so a prompt is not produced.

SavePrincipalComponents2(NumberToSave as Short)

Specify the number of principal components to save, so a prompt is not produced.
`SavePrincipalComponents()` will produce a prompt.

SaveRotatedComponents()

Saves the current rotated components in a new column of the data table.

Spin(pitch As Integer, yaw As Integer, roll As Integer, numTimes As Integer)

Spins the plot with the given `pitch`, `yaw` and `roll`. The plot is spun the number of times specified in the `numTimes` (final) parameter.

SpinPitch(Angle As Integer)

Rotates the plot in the given orientation. The angle that is provided must be between –45 and 45 degrees.

SpinRoll(Angle As Integer)

Rotates the plot in the given orientation. The angle that is provided must be between –45 and 45 degrees.

SpinYaw(Angle As Integer)

Rotates the plot in the given orientation. The angle that is provided must be between –45 and 45 degrees.

StdPrincipalComponents()

This option mirrors its non-automation counterpart, calculating standardized principal components on the launch variables.

Surface Object Methods

The `Surface` object provides a way to manipulate a surface plot.

DisplayOptions(option as surfaceDisplayOptions, flag as Boolean)

Turns on (`True`) or off (`False`) one of more than 15 options related to the display properties of the surface plot. Examples are data points and X Axis Grid.

SetItemColor(item as surfaceColorConstants, color as JMPColorConstants)

Changes the color of a variety of surface plot display elements. Examples are the Mesh and Contour colors.

Survival Object Methods

The `Survival` object provides a way to launch and manipulate a survival / reliability analysis.

CompetingCauseAction(competingCauseConstants action, Flag as Boolean)

Turns on one of several options for the Competing Cause display. A value of `True` for *`Flag`* turns the option on, `False` turns it off.

CompetingCauses(columnName As String) As Boolean

The column parameter is a column in the data table that contains labels for causes of failure. This returns `True` for success, `False` for failure.

ExponentialEst(Flag As Boolean)

Mirrors the non-automation option for plotting. A parameter of `True` turns the option on, `False` turns it off.

ExponentialPlot(Flag As Boolean)

Mirrors the non-automation option for plotting. A parameter of `True` turns the option on, `False` turns it off.

LognormalEst(Flag As Boolean)

Mirrors the non-automation option for plotting. A parameter of `True` turns the option on, `False` turns it off.

LognormalPlot(Flag As Boolean)

Mirrors the non-automation option for plotting. A parameter of `True` turns the option on, `False` turns it off.

MidStepQuantilePoints(Flag As Boolean)

Turns the option on (`True`) or off (`False`).

ReverseYAxis(Flag As Boolean)

This display option mirrors the non-automation `Plot` submenu item. If `Flag` is `True`, the option is turned on, if `False` it is turned off.

SaveEstimates() As DataTable

Creates a new data table that lists the causes of failure. Returns a dispatch pointer to the new data table so it can be manipulated.

ShowCombined(Flag As Boolean)

This display option mirrors the non-automation `Plot` submenu item. If `Flag` is `True`, the option is turned on, if `False` it is turned off.

ShowConfidInterval(Flag As Boolean)

This display option mirrors the non-automation `Plot` submenu item. If `Flag` is `True`, the option is turned on, if `False` it is turned off.

ShowPoints(Flag As Boolean)

This display option mirrors the non-automation `Plot` submenu item. If `Flag` is `True`, the option is turned on, if `False` it is turned off.

SurvivalPlot(Flag As Boolean)

Turns the actual plot on or off.

ShowSimultaneousCI(Flag as Boolean)

Turns the option for these confidence intervals On (`True`) or Off (`False`).

WeibullEst(Flag As Boolean)

Mirrors the non-automation option for plotting. A parameter of `True` turns the option on, `False` turns it off.

Weibull-Plot(Flag As Boolean)

Mirrors the non-automation option for plotting. A parameter of `True` turns the option on, `False` turns it off.

Ternary Object Methods

The `Ternary` object provides a way to launch a ternary plot. It also supports the common analysis automation methods.

LaunchAddFormulaCol(ColumnName As String) As Boolean

Add a column with a contour formula.

LaunchRemoveFormulaCol(ColumnName As String) As Boolean

Remove a column with a contour formula.

Text Explorer

Before Launch()

The following methods need to be called before the Launch() method:

LaunchAddTextColumn(name as String) As Boolean

Assigns the columns that contain text data. `Name` refers to a column name from the data table. `True` is returned upon success and `False` upon a failure.

LaunchAddID(name as String) As Boolean

Assigns a column used to identify separate respondents in the Save Stacked DTM for Association output data table. `Name` refers to a column name from the data table. `True` is returned upon success and `False` upon a failure.

LaunchAddBy(name as String) As Boolean

Identifies a column that creates a report consisting of separate analyses for each level of the variable. `Name` refers to a column name from the data table. `True` is returned upon success and `False` upon a failure.

LaunchMaxWordsPerPhrase(n As Short) As Boolean

Specifies a maximum number of words that a phrase can contain to be included as a phrase in the analysis. `N` refers to the max value. `True` is returned upon success and `False` upon a failure.

LaunchRemoveTextColumn(name As String) As Boolean

Removes a previously assigned Text Column field. `Name` refers to a column name from the data table. `True` is returned upon success and `False` upon a failure.

LaunchRemoveID(name As String) As Boolean

Removes a previously assigned ID field. `Name` refers to a column name from the data table. `True` is returned upon success and `False` upon a failure.

LaunchRemoveBy(name As String) As Boolean

Removes a previously assigned By field. `Name` refers to a column name from the data table. `True` is returned upon success and `False` upon a failure.

LaunchMaxNumberOfPhrases(n As Long) As Boolean

Specifies the max number of phrases that are shown in the phrase list. `N` refers to the max value. `True` is returned upon success and `False` upon a failure.

LaunchMinCharactersPerWord(n As Short) As Boolean

Specifies the number of characters that a word must contain to be included as a term in the analysis. `N` refers to the minimum value. `True` is returned upon success and `False` upon a failure.

LaunchMaxCharactersPerWord(n As Short) As Boolean

Specifies the largest number of characters that a word can contain to be included as a term in the analysis. `N` refers to the maximum value. `True` is returned upon success and `False` upon a failure.

LaunchLanguage(option As textExplorerLangugageOptions) As Boolean -

Specifies the language used for text processing. This option is independent of the language in which JMP is running. Options uses one of the predefined constants for language. `True` is returned upon success and `False` upon a failure.

LaunchStemming(option As textExplorerStemmingOptions) As Boolean

Specifies a method for combining terms with similar beginning characters but different endings. `Options` uses one of the predefined constants for stemming. `True` is returned upon success and `False` upon a failure.

LaunchTokenizing(option As textExplorerTokenizingOptions) As Boolean

Specifies a method for parsing the text into terms or tokens. `Options` uses one of the predefined constants for tokenizing. `True` is returned upon success and `False` upon a failure.

LaunchTreatNumbersAsWords(flag As Boolean) As Boolean

Allows numbers to be tokenized as terms in the analysis. This method only works with the Basic Words tokenizing method. The `Flag` enables or disables the feature. `True` is returned upon success and `False` upon a failure.

Launch() As Boolean

Launches the Text Explorer platform. `True` is returned upon success and `False` upon a failure.

After Launch()

The following methods can be called after the Launch() method has been called:

DisplayOptions(option As textExplorerDisplayOptions, flag as Boolean)

Allows you to enable or disable the various display options associated with a text explorer report. The option refers to one of the `textExplorerDisplayOption` constants. `Flag` specifies whether the option will be enabled or disabled.

LatentClassAnalysis(maxNumTerms As Short, minTermFrequency As Short, numClusters As Short)

Performs a latent class analysis on the binary weighted document term matrix using sparse matrix routines. Requires JMP Pro.

LatentSemanticAnalysis(maxNumTerms As Short, minTermFrequency As Short, weighting As textExplorerSemanticWeightingOptions, numSingularVectors As Short, centeringAndScaling As textExplorerSemanticCenteringOptions)

Performs a sparse singular value decomposition of the document term matrix. Weighting is defined by the `textExplorerSemanticWeightingOptions` constants. `centeringAndScaling` is defined by the `textExplorerSemanticCenteringOptions` constants. Requires JMP Pro.

TopicAnalysis(numTopics As Short)

Performs a varimax rotated singular value decomposition of the document term matrix to produce groups of terms called topics. `numTopics` specifies the number of topics to produce. The method must be run after `Latent Semantic Analysis`. Requires JMP Pro.

ClusterTerms(flag As Boolean)

Shows or hides a hierarchical clustering analysis of the terms in the data. `Flag` specifies whether to enable or disable the feature. Must be run after `Latent Semantic Analysis`. Requires JMP Pro.

ClusterDocuments(flag As Boolean)

Shows or hides a hierarchical clustering analysis of the documents in the data. `Flag` specifies whether to enable or disable to the feature. The method must be run after `Latent Semantic Analysis`. Requires JMP Pro.

SVDScatterplotMatrix(numVectors As Short)

Shows or hides a scatterplot matrix of the term and document singular value decomposition vectors. `numVectors` specifies the number of vectors. The method must be run after `Latent Semantic Analysis`. Requires JMP Pro.

TopicScatterplotMatrix(flag As Boolean)

Shows or hides a scatterplot matrix of the rotated singular value decomposition vectors. `Flag` specifies whether to enable or disable the feature. The method must be run after `Latent Semantic Analysis` and `Topic Analysis`. Requires JMP Pro.

SaveDocumentTermMatrix(maxNumTerms As Short, minTermFrequency As Short, weighting As textExplorereSemanticWeightingOptions)

Saves columns to the data table for each column of the document term matrix. Weighting refers to one of the options from `textExplorerSemanticWeightingOptions` options.

SaveDocumentSingularVectors(numVectors as Short)

Saves a user-specified number of singular vectors from the document singular value decomposition as columns to the data table. The method must be run after `Latent Semantic Analysis`. Requires JMP Pro.

SaveDocumentTopicVectors()

Saves a user-specified number of singular vectors from the rotated singular value decomposition as columns to the data table. Must be run after `Latent Semantic Analysis` and `Topic Analysis`. Requires JMP Pro.

SaveStackedDTMForAssociation() As JMP.DataTable

Saves a stacked version of the document-term matrix to a JMP data table. Returns a reference to the created data table. Requires JMP Pro.

SaveDTMFormula()

Saves a vector-valued formula column to the data table.

SaveSingularVectorFormula()

Saves a vector-valued formula column that contains the document singular value decomposition to the data table. Requires JMP Pro.

SaveTopicVectorFormula()

Saves a vector-valued formula column that contains the rotated singular value decomposition to the data table. The method must be run after `Latent Semantic Analysis`. Requires JMP Pro.

SaveTermTable() As JMP.DataTable

Creates a JMP data table that contains each term from the Term List, the number of occurrences, and the number of documents that contain each term. Returns a reference to the created data table.

SaveTermSingularVectors(numVectors as Short)

Saves a user-specified number of singular vectors from the terms singular value decomposition as columns to the data table saved by the `Save Term Table` command. The method must be run after `Save Term Table`. Requires JMP Pro.

SaveTermTopicVectors()

Saves the topic vectors as columns to the data table created by the `Save Term Table` command. The method must be run after `Save Term Table`. Requires JMP Pro.

ScoreTermsByColumn(columnName As String)

Saves the term list table with scores based on values in a specified column to a JMP data table created by the `Save Term Table` command. `columnName` refers to the name of the specified column. The method must be run after `Save Term Table`.

TimeSeries Object Methods

The TimeSeries object provides a way to launch and manipulate a time series analysis.

ARCoefficients(Flag As Boolean)

This analysis option refers to its non-automation counterpart. A parameter of `True` turns the option on, `False` turns the option off.

Arima(p As Double, d As Double, q As Double, confidenceInterval As Double, intercept As Boolean, constrainFit As Boolean)

Runs an ARIMA model. The parameters mirror those of the ARIMA dialog when running standalone.

Autocorrelation(Flag As Boolean)

This analysis option refers to its non-automation counterpart. A parameter of `True` turns the option on, `False` turns the option off.

ConnectingLines(Flag As Boolean)

Display option for a `TimeSeries` plot. A parameter of `True` turns the option on, `False` turns it off.

MeanLine(Flag As Boolean)

Display option for a `TimeSeries` plot. A parameter of `True` turns the option on, `False` turns it off.

PartialAutocorr(Flag As Boolean)

This analysis option refers to the `Partial Autocorrelation` menu item. A parameter of `True` turns the option on, `False` turns the option off.

SaveSpectralDensity() As DataTable

Saves the spectral density in a new data table, and returns a dispatch pointer to the new table so that it can be manipulated.

ShowPoints(Flag As Boolean)

Display option for a `TimeSeries` plot. A parameter of `True` turns the option on, `False` turns it off.

SmoothingModel(Model As Integer, Constraints As Integer)

Sets the smoothing model and constraints, using values from the `timeSeriesModelConstants` and `timeSeriesConstraintConstants`.

SpectralDensity(Flag As Boolean)

This analysis option refers to its non-automation counterpart. A parameter of `True` turns the option on, `False` turns the option off.

TimeSeriesGraph(Flag As Boolean)

Display option for a `TimeSeries` plot. A parameter of `True` turns the option on, `False` removes the plot entirely.

Variogram(Flag As Boolean)

This analysis option refers to its non-automation counterpart. A parameter of `True` turns the option on, `False` turns the option off.

Variability Object Methods

The `Variability` object provides a way to launch and manipulate a variability chart.

AIAGLabels(Flag As Boolean)

This is a display option controlled by a boolean flag. A `True` means show this option, a `False` means hide it.

BiasReport(Flag As Boolean)

This is a display option controlled by a boolean flag. A `True` means show this option, a `False` means hide it.

ConnectCellMeans(Flag As Boolean)

This is a display option controlled by a boolean flag. A `True` means show this option, a `False` means hide it.

DiscriminationRatio(Flag As Boolean)

This is a display option controlled by a boolean flag. A `True` means show this option, a `False` means hide it.

GageRandR(K As Double, Tolerance As Double)

Mirrors the non-automation counterpart by performing a Gage R&R analysis.

LinearityStudy(Flag As Boolean)

This is a display option controlled by a Boolean flag. A `True` means show this option, a `False` means hide it.

NomAxisBooleanOption(Handle as Long, Action as Short, Flag As Boolean)

Controls a Boolean option for the display of the Nominal Axis. If the platform supports the option, then this method will either turn it on (Flag is `True`) or off (Flag is `False`). Examples of options are Rotated Tick Labels, Divider Bars and displaying a Lower Frame. The Rotated Tick Labels are supported only on the Oneway and Variability Chart platforms. Before this method can be called, a handle to the Nominal Axis display box must be obtained through a call to `GetGraphicItemByType(TypeName As String, ItemNumber As Integer) As Long`.

PointsJittered(Flag As Boolean)

Turns this option on (`True`) or off (`False`).

ShowBoxPlots(Flag as Boolean)

Turns this option on (`True`) or off (`False`).

ShowCellMeans(Flag As Boolean)

This is a display option controlled by a boolean flag. A `True` means show this option, a `False` means hide it.

ShowGrandMean(Flag As Boolean)

This is a display option controlled by a boolean flag. A `True` means show this option, a `False` means hide it.

ShowGroupMeans(Flag As Boolean)

This is a display option controlled by a boolean flag. A `True` means show this option, a `False` means hide it.

ShowPoints(Flag As Boolean)

This is a display option controlled by a boolean flag. A `True` means show this option, a `False` means hide it.

ShowRangeBars(Flag As Boolean)

This is a display option controlled by a boolean flag. A `True` means show this option, a `False` means hide it.

ShowStdDevChart(Flag As Boolean)

This is a display option controlled by a boolean flag. A `True` means show this option, a `False` hides the entire plot.

ShowVariabilityChart(Flag As Boolean)

This is a display option controlled by a boolean flag. A `True` means show this option, a `False` hides the entire plot.

VarianceComponents(option As Integer) As Boolean

Mirrors the non-automation option. The `method` parameter determines what type of statistic to display, and should be a value from the `varVarianceComponentConstants` definition. This method returns `True` for success and `False` for failure.

Technology License Notices

- Scintilla is Copyright © 1998–2017 by Neil Hodgson <neilh@scintilla.org>.

  All Rights Reserved.

  Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.

  NEIL HODGSON DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL NEIL HODGSON BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
- Progressë® Telerik® UI for WPF: Copyright © 2008-2019 Progress Software Corporation. All rights reserved. Usage of the included Progress® Telerik® UI for WPF outside of JMP is not permitted.
- ZLIB Compression Library is Copyright © 1995-2005, Jean-Loup Gailly and Mark Adler.
- Made with Natural Earth. Free vector and raster map data @ naturalearthdata.com.
- Packages is Copyright © 2009–2010, Stéphane Sudre (s.sudre.free.fr). All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

  Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

  Neither the name of the WhiteBox nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
- iODBC software is Copyright © 1995–2006, OpenLink Software Inc and Ke Jin (www.iodbc.org). All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  – Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
  – Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

– Neither the name of OpenLink Software Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL OPENLINK OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- This program, "bzip2", the associated library "libbzip2", and all documentation, are Copyright © 1996–2019 Julian R Seward. All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

  2. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

  3. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

  4. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

  Julian Seward, jseward@acm.org

  bzip2/libbzip2 version 1.0.8 of 13 July 2019
- R software is Copyright © 1999–2012, R Foundation for Statistical Computing.
- MATLAB software is Copyright © 1984-2012, The MathWorks, Inc. Protected by U.S. and international patents. See www.mathworks.com/patents. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.
- libopc is Copyright © 2011, Florian Reuter. All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  – Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and / or other materials provided with the distribution.
- Neither the name of Florian Reuter nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- libxml2 - Except where otherwise noted in the source code (e.g. the files hash.c, list.c and the trio files, which are covered by a similar license but with different Copyright notices) all the files are:

  Copyright © 1998–2003 Daniel Veillard. All Rights Reserved.

  Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

  The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL DANIEL VEILLARD BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

  Except as contained in this notice, the name of Daniel Veillard shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from him.

- Regarding the decompression algorithm used for UNIX files:

  Copyright © 1985, 1986, 1992, 1993

  The Regents of the University of California. All rights reserved.

  THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

  1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

- Snowball is Copyright © 2001, Dr Martin Porter, Copyright © 2002, Richard Boulton.

  All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

  2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

  3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS \"AS IS\" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- Pako is Copyright © 2014–2017 by Vitaly Puzrin and Andrei Tuputcyn.

  Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

  The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

  THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

- HDF5 (Hierarchical Data Format 5) Software Library and Utilities are Copyright 2006–2015 by The HDF Group. NCSA HDF5 (Hierarchical Data Format 5) Software Library and Utilities Copyright 1998-2006 by the Board of Trustees of the University of Illinois. All rights reserved. DISCLAIMER: THIS SOFTWARE IS PROVIDED BY THE HDF GROUP AND THE CONTRIBUTORS “AS IS” WITH NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED. In no event shall The HDF

Group or the Contributors be liable for any damages suffered by the users arising out of the use of this software, even if advised of the possibility of such damage.

- agl-aglfn technology is Copyright © 2002, 2010, 2015 by Adobe Systems Incorporated. All Rights Reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  – Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

  – Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

  – Neither the name of Adobe Systems Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- dmlc/xgboost is Copyright © 2019 SAS Institute.

  Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at

  http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

- libzip is Copyright © 1999–2019 Dieter Baron and Thomas Klausner.

  This file is part of libzip, a library to manipulate ZIP archives. The authors can be contacted at <libzip@nih.at>.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

  2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

  3. The names of the authors may not be used to endorse or promote products derived from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF

MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- OpenNLP 1.5.3, the pre-trained model (version 1.5 of en-parser-chunking.bin), and dmlc/xgboost Version .90 are licensed under the Apache License 2.0 are Copyright © January 2004 by Apache.org.

  You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

  – You must give any other recipients of the Work or Derivative Works a copy of this License; and
  – You must cause any modified files to carry prominent notices stating that You changed the files; and
  – You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
  – If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.
  – You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

- LLVM is Copyright © 2003–2019 by the University of Illinois at Urbana-Champaign.

  Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at:

  http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

- clang is Copyright © 2007–2019 by the University of Illinois at Urbana-Champaign.

  Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at:

  http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS”, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

- lld is Copyright © 2011–2019 by the University of Illinois at Urbana-Champaign.

  Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at:

  http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS”, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

- libcurl is Copyright © 1996–2021, Daniel Stenberg, daniel@haxx.se, and many contributors, see the THANKS file. All rights reserved.

  Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

  THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

  Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

- On the Windows operating system, JMP utilizes the OpenBLAS library. OpenBLAS is licensed under the 3-clause BSD license. Full license text follows: Copyright © 2011-2015, The OpenBLAS Project

  All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

  2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

  3. Neither the name of the OpenBLAS project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OPENBLAS PROJECT OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)

ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.