Automating JMP via Scripting tools and MS Windows COM
Matthew Flynn, The Travelers Co., Hartford, CT

ABSTRACT
One occasionally has a need to automate and control an analysis using JMP on a Windows platform. Perhaps you are a ‘nix SAS® coder with plenty of scripting experience, but are not up to speed with common Windows technologies such as VB, or Visual C++. A standard JMP installation comes with automation examples in those technologies. But beyond those languages, it can be difficult to find examples for scripting JMP on Windows platform. This paper attempts to remedy that situation by providing a walk-through of a JMP scripting session using common external scripting tools. We’ll show and discuss the example using TCL. Appendix A provides parallel code in Tcl, Python, Perl, VBscript, Ruby, and even R. The magic ingredient for this to work so well is MS Windows Component Object Model, or COM.

INTRODUCTION
Dynamic scripting languages provide surprising power and easy of use. A distinct advantage is the use of a common language - across platforms. Today, one can script with one’s favorite scripting language across platforms. On Windows, one key piece to adding powerful control over one’s desktop is the key addition of means to communicate from scripting languages to applications via Component Object Model, or COM. JMP, like SAS/EG and the SAS IOM, fully supports the surfing of its functionality to external tools via the JMP Object model. That functionality can be accessed from common scripting tools with the additional of scripting COM components. This article will show and discuss an example of scripting a JMP application using the scripting Tool Command Language, or Tcl. Appendix A provides complete parallel code for the example in several additional scripting tools: Tcl, Python, Perl, VBscript, Ruby and even R. The magic ingredient for this to work so well is Windows Component Object Model, or COM. So grab your favorite flavor for scripting & off we go!

A code walk-through of some of the key points in the example below will be in TCL. However, full TCL, Python, Perl, VBScript, and Ruby scripts for the example will be attached in the appendix. Key similarities and differences across scripting environments will be noted below.

TCL
To quote the Tcl Wiki[28]:

“Tcl, or the “Tool Command Language”*, is a powerfully simple, open source-licensed programming language. Tcl provides basic language features such as variables, procedures, and control structures. It runs on almost any modern OS, such as Unix (Linux and non-Linux), MacOS, Windows 95/98 (in Tcl’s older forms)/NT/2000/XP/Vista/…, PDA systems, and many more. But the key feature of Tcl is its extensibility. “

One deservedly popular extension to TCL is Tcom[29]:

Tcom is a Windows-specific Tcl extension that provides commands to access and implement COM objects. It enables client-side and server-side scripting of COM objects through IDispatch and IUnknown derived interfaces.

Tcom has been so popular that it is now included in the ActiveState distribution of TCL[30]. One can download and install Tcl on a Windows machine with a couple of clicks, fire up the WISH TCL interpreter and begin scripting within the WISH Console. Tcl scripts are simple text files can be created in your favorite code or text editor. (The pound symbol, “#”, denotes comments). Let’s get started.
# load up the Tcom add-in package
package require tcom
# fire up JMP
set JMP [::tcom::ref createobject "JMP.Application"]; # set a property
$JMP -method Visible 0
# check some properties
puts [$JMP Parent]
puts [$JMP FullName]
puts [$JMP Name]
puts [$JMP Visible]

Here the JMP application object has several properties including whether to display the application with the “Visible” property. There a number of methods available for the application object. Here the JMP VB Documentation help file[1] and an Object browser come in very handy.

**THE CLASSIC – “HELLO WORLD”**

Continuing our example, following the VB example in the JMP Scripting Guide[2], let’s open a new JMP DataTable, populate it with some data, perform some calculations, launch some analysis platforms, write some JSL script. In sort, control everything one might do interactively with JMP!

```tcl
# Call a method
# Make the JMP application visible
$JMP -method Visible 1
# create a new data table
set dt [$JMP NewDataTable {Hello World.jmp}]
# and add a column
set col [$dt NewColumn Col1 1 0 8]; # dtTypeNumeric = 1
# add data to data table via script
for {set i 1} {$i<=20} {incr i} {
    $col SetCellVal $i $i
}
...
set col4 [$dt NewColumn Col3 1 0 8] ;
$col4 AddFormula { Col1 + Col2 + Col3 }
```

Notice that the `::tcom::ref createobject` call creates a COM Object reference that we use invoking the NewDataTable method, create a NEW COM Object we call “dt”, and use to create a new column “col”. A simple “for” loop populates some cells of the datatable. We can also insert JSL formulas, as we do above for column 4. One sticky point is the first parameter after the column name “col1” passed to the NewColumn method call. The “1” represents a column datatype constant for the type “numeric”. How does one determine the “magic number”? It is not in the VB Documentation reference. Here, either an OLE Object browser such as the one within Excel becomes valuable. In TCL, the OPTCL package provides a nice graphical Object browser. Perl has OLE browser as well. With TCL, it is easy to get the required information programmatically. Reading the JMP type library, the “info commands”, and “info vars” list the large number of methods and variables at your disposal. The “parray” command lists the datatype constants.
Tcl Code snippet #3.

::tcom::import "C:/Program Files/SAS/JMP/8/jmp.tlb"
info commands JMP::*
(Produces a large list of available commands…)
info vars JMP::*
(Produces a large list of available variables…)
parray ::JMP::colDataTypeConstants
returns:
::JMP::colDataTypeConstants(dtTypeCharacter) = 2
::JMP::colDataTypeConstants(dtTypeNumeric) = 1
::JMP::colDataTypeConstants(dtTypeRowState) = 3
::JMP::colDataTypeConstants(dtTypeUnknown) = 0

We can continue to programmatically populate our data table, including using TCL arrays, and adding column formulas, see appendix A for examples. Quickly, JMP looks like:

JMP Screen Output #3.

COM communication goes both ways, we have opened JMP, created a new data table, populated that table with data, and performed calculations, now we can easily extract that data back to our calling COM application with the “GetDataVector” command:
One of the dead-easy, simplest ways to quickly develop code is to use the self-recording feature of JMP, interactively run your analysis until you are happy, capture the JSL script. One can now easily run that JSL script from outside of JMP, in your Tcl script with 'RunCommand'. For example:

Tcl Code snippet #5.

```tcl
# Open a sample JMP data table
set JMPdoc [$JMP OpenDocument {C:\Big Class.JMP}]
# Run some JSL
$JMP RunCommand "Distribution(
    Continuous Distribution(
        Column( :weight ),
        Fit Distribution( Normal )
    ),
    Nominal Distribution(
        Column( :age )
    )
)"
```

The Document object has a “CreateOneway” method which can be directly addressed as discussed in the JMP Scripting Guide, Chap 11 - OLE Automation, p.392 using VB. Here it is in Tcl. (Remember "$JMPDoc" is or COM Object representing the open Big Class data table.).
Tcl Code snippet #6.

```tcl
set oneway [JMPdoc CreateOneway]
$oneway LaunchAddY "Height"
$oneway LaunchAddX "Age"
$oneway Quantiles True
$oneway Launch
$oneway MeansAnovaT True
$oneway MeansStdDev True
$oneway UnequalVariances True
$oneway NormalQuantilePlot True
$oneway SetAlpha 0.05
$oneway Save 0
$oneway Save 1
$oneway CompareMeans 1 True
$oneway CompareMeans 0 True
```

JMP Screen Output #6.

Analysis platforms can be launched using JSL or directly creating a platform object from the document.
Tcl Code snippet #7.

```
set JMPdoc [JMP OpenDocument "C:/oilProduction.jmp"]
set bubble [JMP RunCommand "Bubble Plot(
    X( :Year ),
    Y( :Production ),
    Sizes( :Production ),
    Time( :Year ),
    ID( :Country ),
    All Labels( 1 ),
    GO,
    Save as SWF("C:/bubble.wmf")
)"]
```

JMP Screen Output #7.

![Bubble Plot of Production by Year Sized by Production Across Year ID Country](image-url)
WHAT'S NEXT
Now we are "cooking with gas". We can automate JMP to our heart's content. What's next? Perhaps:

- Automating the production of output graphic files.
- Take advantage of the SAS IOM commands and have your JMP script run SAS code, perhaps on a remote server?
- Scheduling your script via Windows -> START -> CONTROL PANEL -> SCHEDULE TASKS.
- Tie your JMP task with other jobs via other schedulers. Scheduling a job to periodically poll a directory for the existence of a new data file to run your analysis.
- Mix and Match with script tools such as Tcl, Python, Perl, or R using each tool most appropriate to the task at hand. Continue COM scripting in something else, perhaps automating the placing JMP output data intro R for more analysis, or the placing of JMP graphics output directly into MS Powerpoint or Excel. They are all COM scriptable as JMP is. And there are a lot more examples available out on the Internet.

COMPARE & CONTRAST
COM Syntax across scripting tools is more alike than different across: Tcl, Perl, Python, R, & VBScript. For example: Start up JMP, Open a Document, Launch a Platform, Run some JSL.

TCL:
```tcl
package require tcom
set JMP [::tcom::ref createobject "JMP.Application"];
set JMPdoc [$JMP OpenDocument {C:\Big Class.JMP}]
set oneway [$JMPdoc CreateOneway]
$JMP RunCommand "Distribution(";
```

Perl:
```perl
use OLE;
use Win32::OLE;
$jmp = Win32::OLE->new('JMP.Application')
my $doc = $jmp->OpenDocument("C:\Big class.jmp");
$doc -> CreateOneway
$jmp->RunCommand("Distribution(....
```

Python:
```python
from win32com.client import Dispatch
jmp = Dispatch("JMP.Application")
doc = jmp.OpenDocument("C:/Big class.jmp")
doc.CreateOneway
jmp.RunCommand("Distribution(...
```
R:
library(rcom)
jmp <- comCreateObject("JMP.Application")
bigclass <- comInvoke(jmp, "OpenDocument", "C:\\Big Class.jmp")
oneway <- comInvoke(bigclass, "CreateOneway")
comInvoke(jmp, "RunCommand",...)

VBScript:
set jmp = Wscript.CreateObject("JMP.Application")
set jmpdoc = jmp.OpenDocument("C:/Big class.jmp")
Set Oneway = JMPDoc.CreateOneway
jmp.RunCommand("Distribution(...

ADDITIONAL RESOURCES

CONCLUSIONS
With JMP and Windows COM and scripting tools, the world’s your oyster. Grab your personal favorite scripting tool, SAS JMP & you’re off to the races.

REFERENCES:
1. VB JMP Object model documentation
   Check your JMP Install directory, mine is located:
   C:\Program Files\SAS\JMP\8\Support Files English\Help\VBDocumentation.chm
2. JMP Scripting Guide,
   HTTP://WWW.JMP.COM/SUPPORT/DOWNLOADS/PDF/JMP8/JMP_SCRIPTING_GUIDE.PDF
3. JMP VB Automation Reference,
   HTTP://WWW.JMP.COM/SUPPORT/DOWNLOADS/DOCUMENTATION.SHTML, and
   HTTP://WWW.JMP.COM/SUPPORT/DOWNLOADS/PDF/VB_AUTOMATION_REFERENCE.PDF
4. Eberhardt, Peter and Richard DeVenezia, Through the looking glass: Two windows into SAS®, SUGI 30, HTTP://WWW2.SAS.COM/PROCEEDINGS/SUGI30/003-30.PDF
12. OLEview (works on Win XP as well) HTTP://WWW.MICROSOFT.COM/DOWNLOADS/DETAILS.ASPX?FAMILYID=5233B70D-D9B2-4CB5-AEB6-45664BE858B6&DISPLAYLANG=EN
17. WIN32::OLE on CPAN, HTTP://SEARCH.CPAN.ORG/~JDB/WIN32-OLE-0.1709/LIB/WIN32/OLE.PM
23. SAS IOM HTTP://SUPPORT.SAS.COM/RND/ITECH/DOC/DIST-OBJ/IOM.H
24. SAS Connecting Clients to Servers, HTTP://SUPPORT.SAS.COM/RND/ITECH/DOC/DIST-OBJ/COMCORBA.HTML
27. TCL, HTTP://WWW.ACTIVESTATE.COM/ACTIVETCL/
28. TCL Wiki, HTTP://WIKI.TCL.TK/
29. Huang, Chin, TCOM HTTP://WWW.VEX.NET/~CTHUANG/TCOM/, HTTP://WIKI.TCL.TK/1821
30. ActiveState TCL, HTTP://WWW.ACTIVESTATE.COM/ACTIVETCL/
ACKNOWLEDGMENTS
SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

Thanks to the good folk @ JMP Tech Support for many helpful discussions!

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

Matt Flynn
Hartford, CT
Phone: (860) 233 - 8562
Email: mattflynn@mac.com

APPENDIX

Complete Example Tcl, Python, Perl, R, Ruby and VB scripts are attached below:

# JMP Automation via COM.Tcl
#---------------------------------------------------------------------------------------
# load an add-on package into TCL to enable easy access to COM calls
# included in the ActiveState TCL Distro   http://www.activestate.com/activeTcl/
# www.vex.net/~cthuang/tcom/
# wiki.Tcl.tk/1821
package require tcom

# fire up JMP
set JMP ::tcom::ref createobject "JMP.Application";
$JMP -method Visible 0

# check current status of value of property
puts [$JMP Parent]
puts [$JMP FullName]
puts [$JMP Name]
puts [$JMP Visible]

# does not work
# $JMP Visible 1
# set interface [tcom::info interface $JMP]
#foreach i $interface {
#  puts $i
#}

# In some cases where tcom cannot get information about the object's interface,
# you may have to use the -method option to specify you want to invoke a method.
# this works - Makes the JMP application visable
$JMP -method Visible 1
puts [$JMP Visible]

# the classic - "Hello World"
# create a new data table
set dt [$JMP NewDataTable {Hello World.jmp}]
set col [dt NewColumn Col1  1  0  8] ;   # dtTypeNumeric=1;
#You must add rows before populating the table with data
$dt AddRows 20 0

# add data to data table via script
for {set i 1} {$i<=20} {incr i} {
    $col SetCellVal $i $i
}
set col2 [dt NewColumn Col2  1  0  8] ; # dtTypeNumeric = 1
for {set i 1} {$i<=20} {incr i} {
    $col2 SetCellVal $i [expr $i + 5]
}
# generate data in a jsl array
for {set i 1} {$i<=20} {incr i} {
    set myArray($i) [expr $i*5]
}
# add data to data table from a jsl array
set col3 [dt NewColumn Col3  1  0  8] ;
for {set i 1} {$i<=20} {incr i} {
    $col3 SetCellVal $i $myArray($i)
}
set col4 [dt NewColumn Col3  1  0  8] ;
$col4 AddFormula { Col1 + Col2 + Col3 }
$dt -method Visible True

# return those JMP calculated values back to TCL
set x [$col4 GetDataVector]

# This adds 5 rows to the end of the table (note error in JMP Scripting quide "0" adds to top of table)
$dt AddRows 5 -1
#This adds 5 rows after row 2
$dt AddRows 5 2
# save the data table using the previously specified file name
set doc [dt Document]
#$doc Save
$doc Close 1 "C://Hello World.jmp"

# Open a sample JMP data table
set JMPdoc [$JMP OpenDocument {C:\Big Class.JMP}]
# It is very easy to grap JSL scripts from JMP and pull them into here.
$JMP RunCommand "Distribution(
    Continuous Distribution( Column( :weight ),
    Fit Distribution( Normal )
),
    Nominal Distribution(
    
),
);"
# The Document object has a CreateOneWay method which can be directly addressed
# As in the JMP Scripting Guide, Chap 11 - OLE Automation, p.392
set oneway [$JMPdoc CreateOneWay]

$oneway LaunchAddY "Height"
$oneway LaunchAddX "Age"
$oneway Quantiles True
$oneway Launch
$oneway MeansAnovaT True
$oneway MeansStdDev True
$oneway UnequalVariances True
$oneway NormalQuantilePlot True
$oneway SetAlpha 0.05
$oneway Save 0; # oscCentered;
$oneway Save 1; # Standardized;
$oneway CompareMeans 1 True; # AllPairs;
$oneway CompareMeans 0 True; # EachPair;

# One note: how to determine the OneWay Save and CompareMeans constants?
# explore the JMP type library a bit
package require tcom
::tcom::import "C:/Program Files/SAS/JMP/8/jmp.tlb"
info commands ::JMP::*
info vars ::JMP::*
# in a vertical (easier to read) list
foreach command [info commands ::JMP::*] {
    puts $command
}
foreach var [info vars ::JMP::*] {
    puts $var
}
parray ::JMP::colDataTypeConstants; # dtTypeCharacter=2, Num=1, RowState=3, Unknown=0;
parray ::JMP::OneWayCompareConstants; # occAllPairs=1, EachPair=0, WithBest=2, WithControl=3;
parray ::JMP::OneWaySaveConstants; # oscCentered=0, NormalQuantiles=2, Standardized=1, Template=2;
parray ::JMP::jmpGraphicsFormats; # jmpGIF=4, jmpJPEG=2, jmpMetaFile=3, jmpPNG=1;

$JMP OpenDocument "C:/automarketshare.jmp"
$JMP RunCommand "Graph Builder(
    Variables(
        X (: Year ),
        Y (: Share ),
        Wrap (: Company ),
        Overlay (: Market )
    )
)"

# platforms can be launched from JSL or directly creating a platform object from the document
set JMPdoc [$JMP OpenDocument "C:/oilProduction.jmp"]
set bubble [$JMP RunCommand "Bubble Plot(
    X (: Year ),
)"
Y( :Production ),
Sizes( :Production ),
Time( :Year ),
ID( :Country ),
All Labels( 1 ),
GO,
Save as SWF("C:/bubble.wmf")
]

$JMP RunCommand "Bubble Plot(Save as SWF('C:/bubble.swf'))"
# "Save as SWF" will pop up a dialog, not sure yet how to hard-code a filename

set JMPdoc [$JMP OpenDocument "C:/oilProduction.jmp"]
set bubble [$JMPdoc CreateBubblePlot]
   $bubble LaunchAddX "Year"
   $bubble LaunchAddY "Production"
   $bubble LaunchAddSizes "Production"
   $bubble LaunchAddTime "Year"
   $bubble LaunchAddID "Country"
   $bubble AllLabels False
   $bubble Launch
   $bubble GO

# $bubble AllLabels True

# save graphic file
$bubble SaveGraphicOutputAs "C:/bubble.png" 1
$bubble SaveGraphicOutputAs "C:/bubble.jpeg" 2
$bubble SaveGraphicOutputAs "C:/bubble.wmf" 3
$bubble SaveGraphicOutputAs "C:/bubble.gif" 4

# save as Shockwave Flash Object
#$bubble Save as SWF("C:/bubble.wmf")

# http://www.jmp.com/support/notes/34/780.html

#################################################
# resources
# VB JMP Object model documentation
# C:\Program Files\SAS\JMP\8\Support Files English\Help\VBDocumentation.chm
# JMP Scripting Guide

# scripting papers
# Boost Your Programming Productivity with SAS® Enterprise Guide®, Chris Hemedinger, SAS,
# www2.sas.com/proceedings/sugi30/019-30.pdf

# use OpTcl to browse the Object model
package require opTcl
# produces big list of registered libraries
typelib::alllibs
foreach name [typelib::alllibs] {
    puts $name
}

# load the JMP library
typelib::load (JMP (Ver 1.0))
typelib::load (RCOM 1.0 Type Library (Ver 1.0))
typelib::load {StatConnectorCommon 1.1 Type Library (Ver 1.1)}
typelib::load {StatConnControls (Ver 9.1)}
typelib::load {StatConnectorCInt 1.0 Type Library (Ver 1.0)}
typelib::load {StatConnectorCommon 1.3 Type Library (Ver 1.3)}

# Open a browser window & drill into the properties and methods
tlview::loadedlibs .w

package require tcom
::tcom::import "C:\WINDOWS\system32\commdlg.dll"
# One can also use the object browser in Excel Macros
# also OLEview (works on Win XP as well)

# JMP Automation via COM.pl

use OLE;
use Win32::OLE;

# define JMP
$jmp = Win32::OLE->new('JMP.Application') or die "oops\n";

# Open existing data table
my $doc = $jmp->OpenDocument("C:\Big class.jmp");
jmp->(Visible)=1;

# the classic
my $dt = $jmp->NewDataTable("Hello World.jmp");
my $col = $dt->NewColumn(Col1, 1, 0, 8);

# You must add rows before populating the table with data
$dt->AddRows(20, 0);

# Set Cell values to increments of 1
for (my $i=1; $i<=20; $i+=1)
{
    $col->SetCellVal($i, $i);
}
my $col2 = $dt->NewColumn(Col2, 1, 0, 8);
for (my $i=1; $i<=20; $i+=1)
{
    $col2->SetCellVal($i,$i + 5);
}
my $col3 =$dt->NewColumn(Col3, 1, 0, 8);
$col3->AddFormula("Col1 + Col2");
$dt->{Visible} = 1;

# return those JMP calculated values back to TCL
my $x = $col3->GetDataVector;

# __ Run some JSL
# $jmp->RunCommand("Distribution(Continuous Distribution( Column( :weight ), Fit Distribution( Normal ) ),
Nominal Distribution( Column( :age ) ) )");

# $jmp->OpenDocument("C:\automarketshare.jmp");
# $jmp->RunCommand("Graph Builder(Variables( X( :Year ), Y( :Share ), Wrap( :Company ), Overlay( :Market ) ) )");

# platforms can be launched from JSL or directly creating a platform object from the document
# my $price = $jmp->OpenDocument("C:\oilProduction.jmp");
# my $bub = $jmp->RunCommand("Bubble Plot( X( :Year ), Y( :Production ), Sizes( :Production ), Time( :Year ),
ID( :Country ), All Labels( 0 ), GO() )");

# platforms can be launched from JSL or directly creating a platform object from the document
# my $price = $jmp->OpenDocument("C:\oilProduction.jmp");
# my $bub = $jmp->RunCommand("Bubble Plot( X( :Year ), Y( :Production ), Sizes( :Production ), Time( :Year ),
ID( :Country ), All Labels( 0 ), GO() )");

# resources
# http://www.activestate.com/activeperl/
# http://search.cpan.org
# http://search.cpan.org/~jdb/Win32-OLE-0.1709/lib/Win32/OLE.pm

# platforms can be launched from JSL or directly creating a platform object from the document
# my $price = $jmp->OpenDocument("C:\oilProduction.jmp");
# my $bub = $jmp->RunCommand("Bubble Plot( X( :Year ), Y( :Production ), Sizes( :Production ), Time( :Year ),
ID( :Country ), All Labels( 0 ), GO() )");

# platforms can be launched from JSL or directly creating a platform object from the document
# my $price = $jmp->OpenDocument("C:\oilProduction.jmp");
# my $bub = $jmp->RunCommand("Bubble Plot( X( :Year ), Y( :Production ), Sizes( :Production ), Time( :Year ),
ID( :Country ), All Labels( 0 ), GO() )");

# platforms can be launched from JSL or directly creating a platform object from the document
# my $price = $jmp->OpenDocument("C:\oilProduction.jmp");
# my $bub = $jmp->RunCommand("Bubble Plot( X( :Year ), Y( :Production ), Sizes( :Production ), Time( :Year ),
ID( :Country ), All Labels( 0 ), GO() )");

# platforms can be launched from JSL or directly creating a platform object from the document
# my $price = $jmp->OpenDocument("C:\oilProduction.jmp");
# my $bub = $jmp->RunCommand("Bubble Plot( X( :Year ), Y( :Production ), Sizes( :Production ), Time( :Year ),
ID( :Country ), All Labels( 0 ), GO() )");
price = jmp.OpenDocument("C:/oilProduction.jmp")
bub = jmp.RunCommand("Bubble Plot( X( :Year ), Y( :Production ), Sizes( :Production ), \
Time( :Year ), ID( :Country ), All Labels( 0 ), GO())")

####################################################
# resources
# http://www.activestate.com/activepython/
# http://docs.python.org/using/windows.html
# http://python.net/crew/mhammond/win32/
# http://www.boddie.org.uk/python/COM.html

####################################################

# JMP Automation via COM.r
####################################################
library(rcom)
# create an instance of JMP
jmp <- comCreateObject("JMP.Application")

# a little fancier
jmpStart = function() {
  if( ! "rcom" %in% .packages() ) library("rcom")
  jmp <- comCreateObject("JMP.Application")
}
jmpStart()

# the application object has some properties
comGetProperty(jmp,"Visible")
# also can be done as:
jmp["Visible"]
jmp["Name"]
jmp["FullName"]
jmp["Parent"]
jmp["Application"]

# we can set some of those properties
comSetProperty(jmp,"Visible",TRUE)
jmp["Visible"] <- TRUE;

comInvoke(jmp, "RunCommand", "jmpversion();")
jmp$RunCommand("jmp version();")
jmp$RunCommand("host is(Windows);")
jmp$GetLogContents();

# let's create a new data table object
jmpdoc <- comInvoke(jmp, "NewDataTable", "Hello World")
comInvoke(jmp, "RunCommand", 'StatusMsg("Hello World")')
# check the new Document's properties
jmpdoc["Document"]
jmpdoc["NumberOfColumns"]
jmpdoc["NumberOfRows"]
jmpdoc["Visible"]
jmpdoc["Visible"] <- "TRUE"
jmpdoc["Visible"]

# some sample data
x <- c(1,2,3,4,5,6)

# now let's populate our JMP data table, adding columns, and rows
addcols <- function(n){
  for (i in 1:n) {
    comInvoke(jmpdoc, "NewColumn", "Column", 1, 0, 10)
    jmpdoc$NewColumn("Column", 1, 0, 10)
  }
}
addcols(3)

fill <- function(n){
  for (i in 1:n) {
    comInvoke(jmpdoc, "GetColumnByIndex", n, "SetDataVector", x)
    comInvoke(jmpdoc$GetColumnByIndex(n), "SetDataVector", x)
  }
}
fill(1)
fill(2)
fill(3)

# we can add a column with a formula
col4 <- comInvoke(jmpdoc, "NewColumn", "Column", 1, 0, 10)
comInvoke(col4, "AddFormula", "Column+Column 2+Column 3")
comInvoke(jmpdoc, "SetWindowSize", "600", "400")
y <- col4$GetDataVector(); y

comInvoke(jmpdoc, "SelectColumn", "Column 4", 1)

# we can see the changed properties of our data table
jmpdoc["NumberOfColumns"]
jmpdoc["NumberOfRows"]

# and we can pass data back to R
y <- comInvoke(col4, "GetDataVector"); y

# when done, we can close the data table
# comInvoke(comGetProperty(jmpdoc, "Document"), "Close", "0", "None")
# comInvoke(jmpdoc["Document"], "Close", "0", "None")
# comInvoke(jmpdoc, "Close", "0", "None")
jmpdoc$Close("0", "None")
bigclass <- comInvoke(jmp, "OpenDocument", "C:\Big Class.jmp")
comInvoke(jmp, "RunCommand",
    "Distribution(
        Continuous Distribution( Column( :weight ),
        Fit Distribution( Normal ) ),
        Nominal Distribution( Column( :age ) ),
    )"
)

# Appendix B, p.559
oneway <- comInvoke(bigclass, "CreateOneway")
    comInvoke(oneway, "LaunchaddX","Age")
    comInvoke(oneway, "LaunchaddY","Height")
    comInvoke(oneway, "Quantiles","True")
    comInvoke(oneway, "Launch")
    comInvoke(oneway, "MeansAnovaT", "True")
    comInvoke(oneway, "MeansStdDev ", "True")
    comInvoke(oneway, "UnequalVariances", "True")
    comInvoke(oneway, "NormalQuantilePlot", "True")
    comInvoke(oneway, "SetAlpha", "0.05")
    comInvoke(oneway, "Save", "Centered")
    comInvoke(oneway, "Save", "Standardized")
    comInvoke(oneway, "CompareMeans", "1", "True"); #occAllPairs
    comInvoke(oneway, "CompareMeans", "0", "True"); #occEachPair

mktshare <- comInvoke(jmp, "OpenDocument", "C:\automarketshare.jmp")
comInvoke(jmp, "RunCommand",
    "Graph Builder(
        Variables(
            X( :Year ),
            Y( :Share ),
            Wrap( :Company ),
            Overlay( :Market )
        )
    )"
)

JMPdoc <- comInvoke(jmp, "OpenDocument", "C:/oilProduction.jmp")
bubble <- comInvoke(JMPdoc, "CreateBubblePlot")
    comInvoke(bubble, "LaunchAddX", "Year")
    comInvoke(bubble, "LaunchAddY", "Production")
    comInvoke(bubble, "LaunchAddSizes", "Production")
    comInvoke(bubble, "LaunchAddTime", "Year")
    comInvoke(bubble, "LaunchAddID", "Country")
    comInvoke(bubble, "AllLabels", "True")
    comInvoke(bubble, "Launch")
    comInvoke(bubble, "GO")

# save graphic file, various formats
comInvoke(bubble, "SaveGraphicOutputAs", "C:/bubble.png", 1)
comInvoke(bubble, "SaveGraphicOutputAs", "C:/bubble.jpeg", 2)
comInvoke(bubble, "SaveGraphicOutputAs", "C:/bubble.wmf", 3)
comInvoke(bubble, "SaveGraphicOutputAs", "C:/bubble.gif", 4)

```
# JMP Automation via COM.vbs
# set jmp = Wscript.CreateObject("JMP.Application")
# jmp.Visible = True

' Appendix B, p.559
set jmpdoc = jmp.OpenDocument("C:/Big class.jmp")

' Launching an analysis
Set Oneway = JMPDoc.CreateOneway
  Oneway.LaunchAddY ("Height")
  Oneway.LaunchAddX ("Age")
  Oneway.Quantiles (True)
  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 1, True
  Oneway.CompareMeans 0, True

jmp.RunCommand("Distribution(Continuous Distribution( Column( :weight ), Fit Distribution( Normal ) ), Nominal Distribution( Column( :age ) ))")

set mktshare = jmp.OpenDocument("C:/automarketshare.jmp")
jmp.RunCommand("Graph Builder(Variables( X( :Year ), Y( :Share ), Wrap( :Company ), Overlay( :Market ) ))")

' platforms can be launched from JSL or directly creating a platform object from the document
set price = jmp.OpenDocument("C:/oilProduction.jmp")
jmp.RunCommand("Bubble Plot( X( :Year ), Y( :Production ), Sizes( :Production ), Time( :Year ), ID( :Country ), All Labels( 0 ), GO() )")
```