Accessing the power & functionality of the MS Windows API via the JMP LoadDll function
Matthew Flynn, The Travelers Co., Hartford, CT

ABSTRACT

This paper will show and discuss JMP scripting code (JSL) examples to access the power & functionality of the MS Windows API via the JMP LoadDll function. Examples include accessing Windows environmental variables, controlling and positioning JMP windows, defining hot-keys, importing and exporting data to MS Excel, GUI automation testing via sending keystrokes. For example, your JMP script can programmatically open an external file with the “ShellExecuteA” function:

shell32 = Load DLL( "shell32" );
shell32 << DeclareFunction("ShellExecuteA",Convention( STDCALL ),Alias( "ShellExecute" ),
   Arg( UIntPtr, input, "hWnd" ),
   Arg( AnsiString( 20 ), "lpOperation" ),
   Arg( AnsiString( 256 ), "lpFile" ),
   Arg( AnsiString( 256 ), "lpParameters" ),/* Arg( UInt32, "lpParameters" ), */
   Arg( UInt32, "lpDirectory" ),
   Arg( UInt32, "nShowCmd" ),
   Returns( UInt32 ) ); //If the function succeeds, it returns a value greater than 32.

// Example call:
fname = "C:\temp\bubble_anno.jpg";

// Fname = "C:\temp\bubble_anno.pdf";
// open in Adobe
rc = shell32 << ShellExecute( 0, "open", fname, " ", 0, 1 );
shell32 << UnloadDLL();

INTRODUCTION

The JMP Scripting guide offers just an example or two to the aspiring JSL programmer to guide in accessing external functions via the LoadDll command. The SAS® Online Docs, however, offer several useful additional examples of common applications of the same functionality with the SAS ModuleN function. The following the JSL scripts follow the SAS examples available under the Online Docs@ Accessing External DLLs from SAS under Windows. There are also plenty of SAS code examples available online, for example, see Richard DeVenezia’s web site, DeVenezia and Loren (2008), Foster (2006), Johnson (2005), and Roper (2001). Here we’ll show examples illustrating several helpful techniques: updating a character string argument, passing arguments by value, and accessing data using a returned pointer, and using structures. Let’s get started.

UPDATING A CHARACTER STRING ARGUMENT

Many Windows functions return only a non-zero number indicating success or, for example, the number of characters in the returned string. The desired output of the function is often returned in an updated parameter of the function. For example, the Windows “GetFullPathNameA” function return code contains the length of the string in the parameter lpBuffer. The output we desire, that is, the directory path name of the file of the current open JMP data table is returned as an updated character string argument. Notice the function requires a pointer to a buffer and also the length of that buffer. We pre-allocate a space for our results with an empty string of length 256 and then call the function.
Define the function with LoadDll.
Declare an empty character buffer to receive the returned string.
The “GetFullPathNameA” function returns the number of characters in the returned string.
Like many Windows functions, our item of interest, the string of the path itself is contained in an updated argument to the main function call (in this case: lpBuffer).

Here’s the documentation from MSDN:


DWORD WINAPI GetFullPathName(
    __in   LPCTSTR lpFileName,
    __in   DWORD nBufferLength,
    __out  LPTSTR lpBuffer,
    __out  LPTSTR *lpFilePart
);

Here’s an example JSL function call:

JSL Code snippet #1.

filepath = Function( {dt},
    If( Num( Substr( JMP Version( ), 1, 3 ) ) == 8,
        kernel32 = Load Dll( "kernel32" );
        MAX_PATH = 255;
        lpBuffer = Repeat( " ", MAX_PATH );
        nBufferLength = Length( lpBuffer );
        lpFilePart = Repeat( " ", MAX_PATH );
        kernel32 << DeclareFunction( "GetFullPathNameA", 
            Convention( STDCALL ), Alias( "GetFullPathName" ),
            Arg( AnsiString( MAX_PATH ), "lpFileName" ),
            Arg( Int32, "nBufferLength" ),
            Arg( AnsiString( MAX_PATH ), update, "lpBuffer" ),
            Arg( Int32, update, "lpFilePart" ),
            Returns( Int32 );
        );// (returns long (the # of characters in the returned string));
        lpFileName = dt << Get Name();
        lpFilePart=0;
        rc = kernel32 << GetFullPathName( lpFileName, nBufferLength,
            lpBuffer, lpFilePart );

        fpath = lpBuffer;
        Show( fpath );
    //kernel32 << ShowFunctions( );
    //kernel32 << UnLoadDll();
    ,
        If( Num( Substr( JMP Version( ), 1, 3 ) ) == 9,
            fpath2 = dt << Get Path(); Show( fpath2 );
        );
    );
    // Example calls
    dt = Open( "$SAMPLE_DATA/Baseball.jmp" );
    filepath(dt);
);
Returns:
rc: 59

fpath:"C:\Program Files\SAS\JMP\8\Support Files English\Sample Data-Sample DataBaseball"

PASSING ARGUMENTS BY VALUE

The simple "Beep" function requires two arguments, frequency and duration, the SAS example at Accessing External DLLs from SAS under Windows, shows passing arguments by value – here we use JSL variables.

- Define the functions with LoadDll.
- Declare JSL variables for frequency and duration.
- Simply call, passing the two input parameters. (Why the fuss? Oddly enough, it won’t work if one tries to pass the values directly).

JSL Code snippet #2.

```javascript
kernel32 = Load Dll( "kernel32" );
dwFreq = 480;
dwDuration = 300;
kernel32 << DeclareFunction("Beep", Convention(STDCALL), Alias("Beep"),
Arg(Int32, input, "dwFreq"),
Arg(Int32, input, "dwDuration"),
Returns(Int32);
); // (returns bool) If the function succeeds, the return value is nonzero.

// example call
rc = kernel32 << Beep(dwFreq, dwDuration);
kernelp32 << UnLoadDll();
```

ACCESSING DATA FROM A RETURNED POINTER

The next example illustrates the use of retrieving a string when function only returns a pointer. Here, “GetCommandLineA” returns a pointer. That is, a location reference in memory. We can use two other helper functions: “lstrlenA” to find out the length of the returned string at the memory location returned by “GetCommandLineA”, and the “RtlMoveMemory” function to copy the chunk of memory at that location and then display it’s value. The steps involve:

- Define the required functions with LoadDll.
- Retrieve pointer to contents of command line with the “GetCommandLineA” function.
- Determine the length of the returned string with “lstrlen”.
- Prep a buffer to receive the copied string.
- Copy the contents of memory at the pointer to the buffer and display.

JSL Code snippet #3.

```javascript
/*
*/
Retrieves the command-line string for the current process

LPTSTR WINAPI GetCommandLine(void);
*/

kernel32 = LoadDll("kernel32");
kernel32 << DeclareFunction("GetCommandLineA", Convention(STDCALL),
Alias("GetCommandLine"),
Returns(Int32));
kernel32 << DeclareFunction("lstrlenA", Convention(STDCALL), Alias("
lstrlen"),
Arg(IntPtr, "Source"),
Returns(Int32));
kernel32 << DeclareFunction("RtlMoveMemory", Convention(STDCALL), Alias("CopyMemory"),
Arg(AnsiString(1024), update, "Destination"),
Arg(IntPtr, "Source"),
Arg(Int32, "Length"),
Returns(Void));
// retrieve pointer to contents of command line
strSource = kernel32 << GetCommandLine();

// returns a pointer, something like 1385232

// determine how long the string is
strSourceLength = kernel32 << lstrlen(strSource);
show(strSource, strSourceLength);

// prep a string buffer (and determine it's length) to receive the value of
// the string
commandLine = Repeat(" ", 255);
commandLength = Length(commandLine);

// copy the contents of memory at the pointer from the location
// of the command line (strSource) to a new string (commandLine) & display
kernel32 << CopyMemory(commandLine, strSource, Min(strSourceLength, commandLength));
commandLine = Trim(commandLine);
Show(commandLine);
kernel32 << UnLoadDll();

Returns:
commandLine:"\\"C:\Program Files\SAS\JMP\8\jmp.exe\\" -N"
USING STRUCTURES

Many Windows functions act on, or return data in flexible data collections known as structures. The next example uses references to display windows – the corner positions of the client window are returned as a logical structure of the left, top, right, and bottom corner positions. We can use these and related window position functions to have complete control over window size and position placement in our application.

- Here, the updated arguments contain the desired data.
- The “lpRect” argument to the function after the call to GetWindowRect contains the four corner positions of the window.
- Another structure in the function “GetSystemMetrics” contains the x and y dimension of the screen.
- We can use the position information to control the position of any window placement on the screen, here, we will center the desired window.

JSL Code snippet #4.

```c
/*
see: JMP Scripting Guide, Version 8, Chapter 12, page 412
Dynamic Link Libraries (DLLs) Windows Only
also:

Using Structures
gle.html
Grouping SAS Variables as Structure Arguments
alDl1topics.htm#structgrp
HWND GetForegroundWindow(VOID);
BOOL GetClientRect(HWND hWnd, LPRECT lprc);

routine GetForegroundWindow
  minarg=0
  maxarg=0
  stackpop=called
  module=USER32
  returns=long;

routine GetClientRect
  minarg=5
  maxarg=5
  stackpop=called
  module=USER32;
arg 1 num input byvalue format=pib4.;
arg 2 num update fdstart format=ib4.;
arg 3 num update format=ib4.;
arg 4 num update format=ib4.;
arg 5 num update format=ib4.;

BOOL GetWindowRect(
  HWND hWnd,
  LPRECT lpRect
)
```
http://support.sas.com/kb/5/538.html
Usage Note 5538: SAS informats that correspond to C data descriptions
The following chart indicates the appropriate SAS informats that correspond to C data types:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Informat</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>RB8.</td>
</tr>
<tr>
<td>float</td>
<td>FLOAT4.</td>
</tr>
<tr>
<td>long</td>
<td>IB4.</td>
</tr>
<tr>
<td>long long</td>
<td>IB8.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>PIB4.</td>
</tr>
<tr>
<td>unsigned long long</td>
<td>PIB8.</td>
</tr>
<tr>
<td>short</td>
<td>IB2.</td>
</tr>
<tr>
<td>char</td>
<td>$CHAR</td>
</tr>
</tbody>
</table>


user32 = Load Dll( "user32" );
user32 << DeclareFunction( "GetForegroundWindow", Convention( STDCALL ),
    Alias( "GetForegroundWindow" ),
    Returns( IntPtr )
); // (args void) (returns long -handle to current foreground window)
 hWnd = user32 << GetForegroundWindow( );

wLeft = wTop = wRight = wBottom = 0;
user32 << DeclareFunction( "GetClientRect", Convention( STDCALL ), Alias( "GetClientRect" ),
    Arg( Int32, input, "hWnd" ),
    StructArg(
        Arg( Int32, "wLeft" ),
        Arg( Int32, "wTop" ),
        Arg( Int32, "wRight" ),
        Arg( Int32, "wBottom" ),
        update, "lpRect" ),
    Returns( Int32 )
);
user32 << GetClientRect( hWnd, wLeft, wTop, wRight, wBottom );
Show( wLeft, wTop, wRight, wBottom );

// gets upper-left window corner position as well as lower-right
// also includes frame
user32 << DeclareFunction( "GetWindowRect", Convention( STDCALL ), Alias( "GetWindowRect" ),
    Arg( Int32, input, "hWnd" ),
    StructArg(
        Arg( Int32, "wLeft" ),
        Arg( Int32, "wTop" ),
        Arg( Int32, "wRight" ),
        Arg( Int32, "wBottom" ),
        update, "lpRect" ),
    Returns( Int32 )
);
user32 << GetWindowRect( hWnd, wLeft, wTop, wRight, wBottom );
Show( wLeft, wTop, wRight, wBottom );

/*
BOOL SetWindowPos(
    HWND hWnd, 
    HWND hWndInsertAfter, 
    int X, 
    int Y, 
    int cx, 
    int cy, 
    UINT uFlags;
*/

user32 << DeclareFunction( "GetSystemMetrics", Convention( STDCALL ), Alias( "GetSystemMetrics" ),
    Arg( Int32, "nIndex" ),
    Returns( Int32 ) );

SM_CXSCREEN = 0; // X Size of screen
SM_CYSCREEN = 1; // Y Size of Screen
xscreen = user32 << GetSystemMetrics( SM_CXSCREEN ); show( xscreen );
    // 1280
    Print("X Size of screen:" || char( xscreen ) );
yscreen = user32 << GetSystemMetrics( SM_CYSCREEN ); show( yscreen );
    // 800
    Print("y Size of screen:" || char( yscreen ) );

SM_CXDRAG = 68;
SM_CYDRAG = 69;
xdrag = user32 << GetSystemMetrics( SM_CXDRAG ); show( xdrag );
    // Print("X Drag:" || char( xdrag ) );
ydrag = user32 << GetSystemMetrics( SM_CYDRAG ); show( ydrag );
    // Print("y drag:" || char( ydrag ) );

x = (xscreen - (wRight - wLeft))/2;
y = (yscreen - (wBottom - wTop))/2;
cx = (wRight - wLeft);
cy = (wBottom - wTop);
user32 << DeclareFunction( "SetWindowPos", Convention( STDCALL ), Alias( "SetWindowPos" ),
    Arg( Int32, input, "hWnd" ),
    Arg( Int32, input, "hWndInsertAfter" ),
    Arg( Int32, input, "x" ),
    Arg( Int32, input, "y" ),
    Arg( Int32, input, "cx" ),
    Arg( Int32, input,"cy" ),
    Arg( Uint32, "uFLAGS" ),
    Returns( Int32 ) );
/* x = y = 80; cx = 500; cy = 600; */
HWND_TOPMOST = -1;
HWND_NOTOPMOST = -2;
SWP_NOSIZE = hex to number("00000001");
SWP_NOMOVE = hex to number("00000002");
SWP_NOACTIVATE = hex to number("00000010");
SWP_SHOWWINDOW = hex to number("00000040");
uFLAGS= SWP_SHOWWINDOW;

// topmost and centered
user32 << SetWindowPos( hWnd, HWND_NOTOPMOST, x, y, cx, cy, SWP_SHOWWINDOW );

// center
user32 << DeclareFunction( "MoveWindow", Convention( STDCALL ), Alias( "MoveWindow" ),
    Arg( Int32, input, "hWnd" ),
    Arg( Int32, "X" ),
    Arg( Int32, "Y" ),
    Arg( Int32, "nWidth" ),
    Arg( Int32, "nHeight" ),
    Arg( Int8, "bRepaint" ),
    Returns( Int8 ) );
bRepaint = 1;
user32 << MoveWindow( hWnd, X, Y, (wRight - wLeft), (wBottom - wTop), bRepaint );

/*
BOOL MoveWindow( HWND hWnd,
    int X,
    int Y,
    int nWidth,
    int nHeight,
    BOOL bRepaint
);
*/

x = 30;
y = 30;
nWidth = 1000;
nHeight = 800;
bRepaint = 1;
user32 = Load Dll( "user32" );
user32 << DeclareFunction( "MoveWindow", Convention( STDCALL ), Alias( "MoveWindow" ),
    Arg( Int32, input, "hWnd" ),
    Arg( Int32, "X" ),
    Arg( Int32, "Y" ),
    Arg( Int32, "nWidth" ),
    Arg( Int32, "nHeight" ),
    Arg( Int8, "bRepaint" ),
    Returns( Int8 ) );

user32 << DeclareFunction( "GetClientRect", Convention( STDCALL ), Alias( "GetClientRect" ),
    Arg( Int32, input, "hWnd" ),
    StructArg(
        Arg( Int32, "wLeft" ),
        Arg( Int32, "wRight" ),
        Arg( Int32, "wTop" ),
        Arg( Int32, "wBottom" )));
Arg( Int32, "wTop" ),
Arg( Int32, "wRight" ),
Arg( Int32, "wBottom" ),
    update, "lpRect" ),
Returns( Int32 )
);
user32 << GetClientRect( hWnd, wLeft, wTop, wRight, wBottom );
Show( wLeft, wTop, wRight, wBottom );

user32 << MoveWindow( hWnd, X, Y, nWidth, nHeight, bRepaint );
Show( X, Y, nWidth, nHeight, bRepaint );
user32 << GetClientRect( hWnd, wLeft, wTop, wRight, wBottom );
Show( wLeft, wTop, wRight, wBottom );

user32 << UnloadDll( );

The above example returns:

    wLeft:0
    wTop:0
    wRight:1262
    wBottom:897
    X:30
    Y:30
    nWidth:1000
    nHeight:800
    bRepaint:1
    wLeft:0
    wTop:0
    wRight:992
    wBottom:750

We can now programmically position and move and dialogs or windows. Another related example was recently posted on the JMP File Exchange, “How to get monitor Resolution”, by Woo-Seaq Je. To get to the File Exchange, start at the JMP home page, www.jmp.com, select User Community from the Blue title bar, and the left-hand box offers a link to “browse the File Exchange”. There one will find a great collection of JMP examples to browse and try.

A MORE COMPLEX EXAMPLE
Let’s go out to the Windows environment and grab a list of all global environmental variables and bring them into JMP as a data table and then display a dialog to allow the user to select a variable. Once selected, we’ll display it’s current value and set the environmental variable to a global JSL variable. The array and display box code was inspired by examples from a great guide that should be on every JMP scripter’s bookshelf, Jump into JMP Scripting, by Wendy Murphy and Rosemary Lucas.

JSL Code snippet #5.
/*
LPTCH WINAPI GetEnvironmentStrings(void);
void CopyMemory(
    __in  PVOID Destination,
    __in  const VOID *Source,
    __in  SIZE_T Length
);

BOOL WINAPI FreeEnvironmentStrings(
    __in  LPTCH lpszEnvironmentBlock   //A pointer to a block of
    //environment strings. The pointer to the block must be obtained by calling
    //the GetEnvironmentStrings function.
);  //If the function succeeds, the return value is nonzero.
    //If the function fails, the return value is zero To get extended er-
    //ror information, call GetLastError.

DWORD WINAPI GetEnvironmentVariable(
    __in_opt   LPCTSTR lpName,
    __out_opt  LPTSTR lpBuffer,
    __in       DWORD nSize
);

    // note env strings contain embedded nulls, eg.
Var1=Value1\0
Var2=Value2\0
Var3=Value3\0
...
VarN=ValueN\0\0

int lstrlen(
    LPCTSTR lpString   //[in] Pointer to a null-terminated string.
);  //The return value specifies the length of the string, in TCHAR
    //values. This refers to bytes for ANSI versions of the function or WCHAR val-
    //ues for Unicode versions. If lpString is NULL, the return value is 0.
    // lstrlen assumes that lpString is a null-terminated string, or NULL.
*/

kernel32 = Load Dll( "kernel32" );
kern32 << DeclareFunction( "GetEnvironmentStringsA", Convention( STDCALL ), Alias( "GetEnvironmentStrings" ),
    Returns( UIntPtr ) );
kern32 << DeclareFunction( "lstrlenA", Convention( STDCALL ), Alias( "lstrlen" ),
    Arg( IntPtr, "Source" ),
    Returns( Int32 ) );
kern32 << DeclareFunction( "RtlMoveMemory", Convention( STDCALL ), Alias( "CopyMemory" ),
    Arg( AnsiString( 2048 ), update, "Destination" ),
    Arg( IntPtr, "Source" ),
    Arg( Int32, "Length" ),
    Returns( Void ) );
kernel32 << DeclareFunction("FreeEnvironmentStringsA", Convention( STDCALL ), Alias("FreeEnvironmentStrings"),
    Arg( IntPtr, "strSource" ),
    Returns( Int32 ) );

strSource = 0;
// retrieve the initial pointer to the environment strings
strSource = kernel32 << GetEnvironmentStrings();
Show( strSource );
strSourceLength = kernel32 << lstrlen( strSource ); show( strSourceLength );
maxStrSourceLength = strSourceLength;
env = Associative Array();
While( strSourceLength > 0,
    // copy the text from the Enviromental block
    strDestination = Repeat( " ", 2048 ); //my path is 1069
    rc = kernel32 << CopyMemory( strDestination, strSource, strSourceLength );
    // strip out trailing white space
    strDestination = Trim( strDestination );
    //Show( strDestination );
    var = substr( strDestination, 1, contains( strDestination, "=" ) - 1 );
    val = substr( strDestination, contains( strDestination, "=" ) + 1, Length( strDestination ) );
    // Show( var, val );
    env[ EvalInsert( "^var^" ) ] = val;
    // move the pointer to the next item
    strSource = strSource + strSourceLength + 1;
    strSourceLength = kernel32 << lstrlen( strSource );
    if( strSourceLength > maxStrSourceLength,
        maxStrSourceLength = strSourceLength );
);

// clean up
Show( maxStrSourceLength );
rc = kernel32 << FreeEnvironmentStrings( strSource );
Show( rc );
kernel32 << UnloadDll( );

// some array functions
total = N Items( env );
env << First;
env << get contents;
Contains( env, "windir" ); // yes
Contains( env, "xyz" ); // no
envvars = env << get keys;
envvals = env << get values;
env << get values( { "TEMP", "TMP" } );
env << get value("windir");
env << get value("Path");
Show( Length(env << get value("Path") );

dt = new table( "env", invisible );
dt << new column("envVar", Character);
dt << new column("envVal", Character);
// iterate through the list/array
for (i = 1, i <= total, i++){
  envVar[i] = envvars[i];
  envVal[i] = envvals[i];
}
// make visible
dt << New Data View();
// Curiously, data tables "Visible" property are not scriptable in JSL,
// here is a way to recover by creating a new table
// one could also go out to script the COM object.
//x = dt << Get Script; x;
//Close( dt );

/*-------------------------------------------------------------------*/
/*       Jump into JMP(r) Scripting                                  */
/* http://www.sas.com/apps/pubscat/bookdetails.jsp?catid=1&pc=61733  */
/* Solution 6.8 */
/*-------------------------------------------------------------------*/
/* Create lists of the values found in the Manufacturer and Model columns. */
Summarize( envVarlist = By( Column( dt, "envVar" ) ) );
Summarize( envVallist = By( Column( dt, "envVal" ) ) );

/* Use New Window to create a non-modal dialog to display var and valfor
  user to choose. */
nw = New Window( "Choose Environmental Variable",
  hb = H List Box( panelBox( "Select an Environmental Variable",
    select1 = Combo Box( envVarlist,
      /* Each of the following tasks are executed when the user makes a Manufac-
         turer choice.*/
      dt << Select Where( :envVar == envVarlist[select1 << get] );
      var = envVarlist[select1 << get];
      selRows = dt << Get Selected Rows;
      pb << delete;
      myVals = {};
      /* The myModels list is populated with values from the selected manufacturer
       models. */
      For( i = 1, i <= N Row( selRows ), i++,
        Insert Into( myVals, Column( dt, "envVal" )[selRows[i]]
      )
    );
    /* A new panel box is drawn and values shown are based on manufacturer se-
     lection. */
    hb << Append( pb = Panel Box( "Environmental Variable Value", select2 = List Box( myVals, width(450) ) ) );
    val = myVals;
  )
),
  pb = Panel Box( "Environmental Variable Value", Combo Box( " " )
),
}
/* If user clicks OK, values displayed for manufacturer and model will be selected in table. */
Button Box("OK",
    nw << Close Window;
    dt << Select Where(
        :envVar == envVarlist[select1 << get];// & :envVal == myVals;
    );
/* Selected rows are subsetted into new table. */
//dt << Subset( Output Table Name("Selected Variable(s)" ) );
user32 = Load Dll("user32") ;
user32 << DeclareFunction("MessageBoxA", Convention( STDCALL ),
    Alias( "MsgBox" ),
    Arg( IntPtr, "hWnd" ),
    Arg( AnsiString( 256 ), "lpText" ),
    Arg( AnsiString( 256 ), "lpCaption" ),
    Arg( UInt32, "uType" ),
    Returns( Int32 )
    );
rc = user32 << MsgBox( 0, EvalInsert("^var^ ||" = " || "^val^"),
    "Environmental Variable", 64 );
user32 << UnloadDll( );
}
resultString = Eval Insert("^var^ = ^val^;");
show( var, val );
Show(dt);
kernel32 << UnloadDll();

// Click OK

**ADDITIONAL RESOURCES**
There are a number of excellent SAS papers and examples available only. The JMP LoadDll function makes it easy to translate SAS MODULEN examples to JSL. See references below.

**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:
5. Hiemstra, Stephen W. and Robert M. Anderson, Calling C libraries from PC SAS and SAS Unix, [HTTP://MEMBERS.COX.NET/MHIEMSTRA/C_WITH_SAS.PDF](HTTP://MEMBERS.COX.NET/MHIEMSTRA/C_WITH_SAS.PDF)

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
Email: mattflynn@mac.com