**PharmaSUG 2012 - Paper CC20**

# Beyond %LET: Building a GUI to Pass Run-Time Parameters to SAS Programs Using VBScript and HTML Applications

Shawn Hopkins, Seattle Genetics, Bothell, WA

## ABSTRACT

Changing a program in a validated environment requires review and documentation. Maintaining validation documentation for programs that require changes to macro variable assignments or LIBNAMEs with each submission can become tedious. There are also the additional risks of users making unintentional changes to the source code, or providing unexpected values to macro parameters. Accepting specifications through a Graphic User Interface (GUI) at run-time allows the program both the flexibility to dynamically modify parameters within the code, and ensure only valid values are provided for parameters. Using VBScript and VBScript within an HTML application (HTA) provides functionality to collect user input and pass the values to SAS® programming environment. This approach leverages simple VBScript functions, the familiar file dialog boxes used in Microsoft Office applications, and allows custom form creation with a variety of controls to meet most business needs.

## INTRODUCTION

Developing dynamic applications dependent on user input can be accomplished in a variety of ways. The simplest method is to assign the values to macro parameters or global macro variables. However, if the end-user is not a programmer, editing SAS code may not be intuitive. A more robust approach is to provide GUI that the user can interact with instead of editing code. This will eliminate accidental code modifications, provide intuitive controls for the end-user, and allow for validation of the parameters prior to executing the SAS code.

This paper will demonstrate how to create increasingly powerful GUI applications using technologies available on a Windows PC with SAS9 and Microsoft Office 2003 or later installed.

## INTERACTIVE DIALOGS WITHIN A SAS PROGRAM

Within a single DATA step, a VBScript can be created in the WORK folder of the SAS instance, executed, and return the results to the data step. VBScript functions and the MS FileDialog object can be used to

- Prompt the user to provide a default reply (Yes, No, OK, Cancel, Quit)
- Enter and validate text strings
- Select one or more files or folders
- Apply conditional logic based on the user response

### SIMPLE GUI USING THE DATA STEP

The DATA step has the same format regardless of the VBScript code generated. Only the code written to the VBS file with the PUT statement will vary.

Creating the data step

- Create an external .vbs file using the FILE statement with FILEVAR option in the WORK folder (Generating the script in the WORK folder automates the removal of the files once the SAS job is complete)
- Write the VBScript into the file using the PUT statement
- Close the .vbs file by reassigning the FILEREF used to create it to a temporary file
- Pass the command to execute the script to FILEVAR option on an INFILE statement
- Read the lines generated from the script from the output stream into the data step using the INPUT statement
- Parse the value of the automatic variable _INFILE_

```
DATA _NULL_;
/* Open the destination VBScript file*/
  vbscript = catx('\',pathname('WORK'),'example.vbs');
  filvar=vbscript;
  file vbs filevar=filvar lrecl=2000;
/* Write the VBScript code into the external file example.vbs using the PUT
statement*/
  put " Replace with the example VBScripts below";
/* Close example.vbs by assigning another file to the fileref VBS */
  filvar = catx('\',pathname('WORK'),'dummyFile.txt');
  file vbs filevar=filvar;
/* Pass the  command to execute the VBScript to FILEVAR statement */
  runvb = 'cscript //nologo '||quote(vbscript);
  infile retval pipe filevar=runvb lrecl=32000 end=eof;
/* PIPE results from the output stream to the DATA _NULL_ step, which can then be
read from the automatic variable _INFILE_, parsed, and stored in macro variables */
  input;
/* Parse the value of _INFILE_ and store in a macro variable for use in the
program*/
  call symputx("parameter1", scan(_INFILE_,1, "<delimiter>");
run;
```

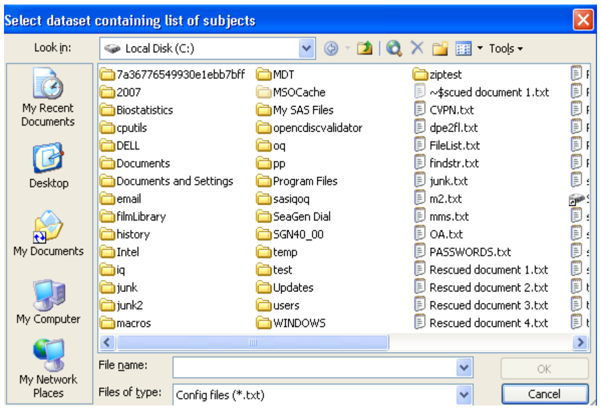
### VBScript Example: Using VBScript functions in the DATA Step

This example asks the user a Yes/No question. If the response is "Yes", the script prompts the user for a file name until a value is provided. The “ECHO” method outputs pipe-delimited values to the console window. The values of archiveYN and archiveName are returned to the data step using the INPUT statement. The _INFILE_ automatic variable is parsed using the SCAN function with “|” delimiter to capture the user responses.

```
 put 'archiveYN=msgBox("Archive existing data before copying new
   files?",36,"Example")'/
      'if archiveYN=6 then'/
      '  while len(archiveName)<1'/
      '    archiveName=inputBox ("Enter a name for the .zip file")'/
      '  wend'/
      'end if'/
      'wscript.echo archiveYN & "|" & archiveName';
```

### VBScript Example: Using the MS Word application object in the DATA Step

VBScript does not offer much in the way of graphic interfaces. The Microsoft Word application object provides functionality to create graphic file dialog boxes. The FileDialog object allows the presentation of a familiar file dialog box, and prompts the user to select one or more files or folders. Though the properties can be modified to suit the application, the appearance of the dialog box cannot be modified. The following code, when inserted into the DATA _NULL_ code above, generates the dialog box in Figure 1.



**Display 1: File dialog box**

```
put 'getFileName' /
      'sub getFileName' /
/* Create the word application Object */
```

```
      'set oWord = CreateObject("Word.Application") '/
/* Allow user to select files only */
      'msoFileDialogFilePicker = 3'/
/* Set the initial folder to C: */
      'oWord.ChangeFileOpenDirectory("C:\")'/
/* Select the dialogType (File picker) */
      'set fd = oword.FileDialog(msoFileDialogFilePicker)  '/
/* Set the properties */
         'with fd   '/
        '.title="Select data set containing list of subjects"'/
        '.AllowMultiSelect = false    '/
        '.Filters.Add "SAS datasets", "*.sas7bdat", 1'/
/* Use the SHOW method to display the dialog, If the user quits without selecting a
file, then closed the object */
        'if .show<>-1 then'/
          'oWord.Quit   '/
          'set oWord = Nothing  '/
          'exit sub'/
        'end if'/
      'end with'/
/* otherwise, assign the value of the selected file to a variable */
      'selectedFile=oword.fileDialog(1).selectedItems(1) '/
      'oWord.Quit   '/
      'set oWord = Nothing'/
/* Echo the selected file to the console */
      'wscript.echo selectedFile' /
      'end sub';
```

## THE HTML APPLICATION

The biggest limitation to using VBScript functions and Word application objects as GUIs are the fact they can't be customized. Developers need more flexibility to be able to create custom forms with a variety of controls, like those found on an interactive webpage. HTAs provide support for HTML, cascading style sheets (CSS), and scripting languages. Plus HTAs execute in the mshta.exe process as trusted applications, disabling the security warnings that would occur with applications run in iexplorer.exe process.

The format of an HTA is similar to a Dynamic HTML (DHTML) page.

Tags used in HTAs

- HTA – contains the attributes that identify the file as an HTML application and defines how the application will behave when started. The HTA tag is located within the HEAD tag
- STYLE – contains the CSS rules that describe the presentation of the HTML.
- SCRIPT – contains the core functionality of the application. In this example, the event-driven VBScript routines and functions are defined here. Scripts can also reside in the BODY tag (see ADaM radio button later)
- BODY – contains the markup content.

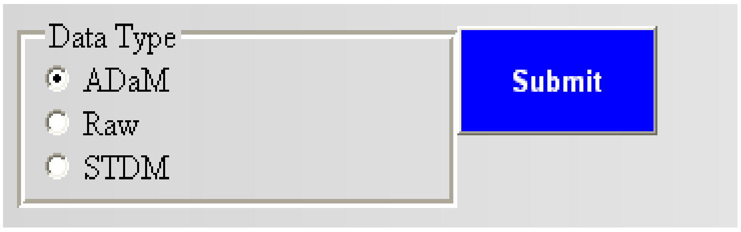
HTML elements alone do not provide the kind of functionality needed to create dynamic, data-driven applications. The collective technologies of a scripting language such as VBScript, CSS, and HTML allow programmers to rapidly develop custom GUIs.

```
<HTML>
<HEAD>
<TITLE>Example</TITLE>
<HTA:APPLICATION
     ID="objHTA"
     APPLICATIONNAME="htaApp "
     SCROLL="yes"
     SINGLEINSTANCE="yes"
     WINDOWSTATE="normal">

<STYLE type="text/css">
</STYLE>
```

```
<SCRIPT Language="VBScript">
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

## DEVELOPING AN HTA APPLICATION

In this example, style, script, and HTML markup is added to the tags in Display 2. The example HTA contains a radio button control that will identify RAW, ADaM, or SDTM. A submit button, which is only visible once a radio button is checked, will batch submit a SAS macro. The value of the checked radio button is passed to the SAS program environment via the SYSPARM option.



**Display 2: Radio and button controls**

### Create styles for the HTML elements

CSS is a style sheet language that defines the presentation of the HTML elements. The syntax consists of a selector (element type, ID, or class) followed by a brace-enclosed list of property ":" value pairs delimited by semicolons. CSS allows the developer to control the appearance, position, and flow of HTML elements.

```
<STYLE type="text/css">
/* Hide the submit button. (Referenced by its ID) */
  #submit{display: none; }
/* Define the style of the FIELDSET element */
  fieldset { width: 220px; border: double; border-style: groove;}
/* Create a class for BUTTON elements */
  .button1 {background-color: blue; color: white;}
</STYLE>
```

### Provide functionality to HTML elements using VBScript

The script tag contains the core logic that will provide the event-driven functionality. In this example there are two subroutines defined, displayElement and runSAS.

*displayElement* accepts the element ID in the parameter "element". When any of the radio buttons are clicked, the display style of the object is set from "none" to "block" using the Document object.

*runSAS* accepts two parameters, SAS program name in sasPgm, and the program location in pgmPath. It loops through the radio buttons, identifies which is checked, and stores that value in the variable sysparmVal. A SAS batch submit command is created with the value of sysparmVal provided to the SYSPARM command line option. The SAS program is submitted using the RUN method of the WSCRIPT.SHELL object.

```
<script language="VBScript">
'Subroutine to set the Submit button to visible
  sub displayElement(Element)
    document.getElementByID(element).style.display="block"
  end sub
'Subroutine to submit SAS program
  sub runSAS(sasPgm,pgmPath)
    set wso=CreateObject("WScript.Shell")
'Identify which radio button was selected and store in variable sysparmVal
    arrRadio=array("adam","raw","stdm")
    for i=0 to ubound(arrRadio)
      if document.getElementByID(arrRadio(i)).checked=True then
        sysparmVal=document.getElementByID(arrRadio(i)).value
      end if
    next
'Populate the options for batch submit command
    sasEXE="C:\Program Files\SAS\SASFoundation\9.2\sas.exe"
    sasCFG="C:\Program Files\SAS\SASFoundation\9.2\sasv9.cfg"
    sasLog=pgmPath   & "\" & sasPgm & ".log"
    sasPrint=pgmPath & "\" & sasPgm & ".lst"
    sasPgm=pgmPath   & "\" & sasPgm & ".sas"
' Uncomment the following line after the dynamic content below is added
' sysparmVal=sysparmVal & " | " &  listSelSub
'Build batch job command line
    cmd=" -sysin " & Chr(34) & sasPgm     & Chr(34) & _
    " -log "       & Chr(34) & sasLog     & Chr(34) & _
    " -print "     & Chr(34) & sasPrint   & Chr(34) & _
    " -config "    & Chr(34) & sasCFG     & Chr(34) & _
    " -sysparm "   & Chr(34) & sysparmVal & chr(34)
'Batch submit sas program using wscript.shell RUN method
    wso.run Chr(34) & sasEXE & Chr(34)  & cmd, 1, false
    window.close
  end sub
</script>
</head>
```

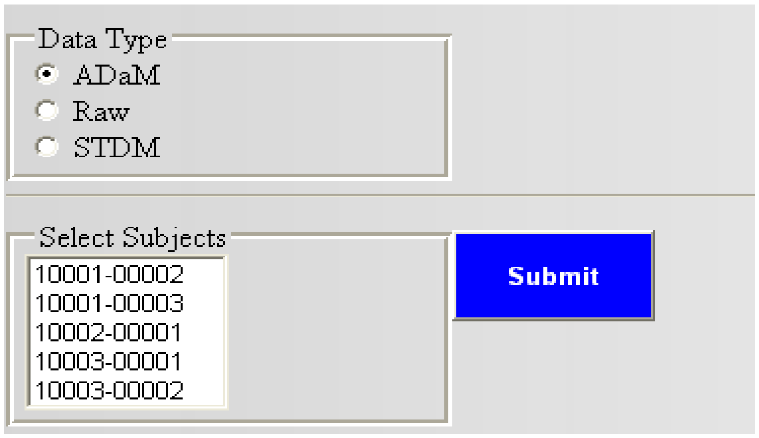
### Define the HTML elements

The body contains the definition of the elements that will be displayed on the page. There are two controls on the page: a collection of radio buttons, and a submit button. For added readability, the radio buttons are enclosed within a “Data type” FIELDSET tag. The displayElement subroutine is associated with the radio button’s ONCLICK event, and the runSAS subroutine is associated with the Submit button’s ONCLICK event. The ADaM radio button has its ONCLICK event defined in the body, though it has the same functionality as the displayElement subroutine.

```
<BODY>
<FIELDSET>
  <LEGEND>Data type</LEGEND>

<INPUT type="radio" id="adam" name="dt" value="adam" >
<LABEL for="adam">ADaM</LABEL>
<!--Scripts can also be written in the body and attached to an event -->
<SCRIPT for="adam" event="onClick"
language="VBScript">document.getElementByID("Submit").style.display="block"
</SCRIPT><BR/>
<!—Subroutine displayElement called from the OnClick event -->
<INPUT type="radio" id="raw" name="dt" value="raw"
onclick="displayElement('submit')">
<LABEL for="raw">Raw</LABEL></BR>

<INPUT type="radio" id="stdm" name="dt" value="stdm"
onclick="displayElement('submit')">
  <LABEL for="stdm">STDM</LABEL>
</FIELDSET>

<!—Precede calls to subroutines with >1 parameter with ‘call’  -->
<INPUT type="button" class="button1" id="submit" value="Submit" onclick="call
runSAS('mcrsas','C:\Documents and Settings')" >
</BODY>
</HTML>
```

## ADDING DYNAMIC ELEMENTS TO THE HTML APPLICATION

Developing robust GUI applications requires data-driven controls. In cases where the options for an element reside outside of the HTA, VBScript’s ability to instantiate Microsoft’s Component Object Model (COM) Objects can be leveraged to access the functionality of other applications. Using a scripting language, dynamic controls can be populated with values returned from a SAS data set. In this example, the ‘Select Subjects’ SELECT element options are values of the USUBJID variable in a SAS data set.



**Display 3: Dynamic select element**

**Add code to the SCRIPT tag**

The dynamic drop-down menu in Display 3 can be created and used by following three steps*:*

- Get the list of values of UBSUBJID from a data set: getSubjects function
- Populate the options of a SELECT element with those values: popSelect subroutine
- Retrieve the selected values from the SELECT element: listSelSubs function

These steps can be accomplished by creating the functions and subroutine below.

*GetSubjects:* This function accepts the path of the LIBREF and the data set name. It executes the code in sqlStmt using the COM to access the Integrated Object Model (IOM) LanguageService to generate the list output. Using the SCRIPTO component, the flushListLines method captures the list output into three arrays. The array containing the values of USUBJID is parsed, stored in a separate array, and assigned to the return variable.

*PopSelect:* This function accepts the name of the Select element and populates the options with the values of subject returned from the getSubjects function.

*ListSelSubs:* This subroutine is called within the runSAS subroutine to add the comma-delimited values of the selected subjects in the “Select Subject” SELECT element to the sysparmVals variable. This is passed to the SAS program via the SYSPARM option.

NOTE: Add “sysparmVal=sysparmVal & " | " & listSelSub“ to the runSAS subroutine prior to the assignment of the CMD variable.

```
function getSubjects(libref,dsn)
'Execute SAS code to create LST output of containing desired variable
sqlStmt="libname in '" & libref & "'; options ps=10000 nodate nonumber;" & _
"PROC SQL;select usubjid from in." & dsn & " order by subject;quit;"
    set obSAS = CreateObject("SAS.Workspace.1.0")
    obSAS.LanguageService.Submit sqlStmt
' Use IOM interface from VBSCRIPT to capture LST output from SQL PROC
    set obScripto = CreateObject("SASScripto.Scripto")
    obScripto.SetInterface obSAS.LanguageService
    Dim fl(3)
    fl(3) = 10000
    obScripto.InvokeMethod "FlushListLines", fl
    for i=0 to ubound(fl (2))
      if inStr(fl(0) (i),"-") > 0 then 'Identify LST line as a subject
        subjects=trim(subjects) & " " & trim(fl(0) (i))
      end if
    next
    getSubjects=split(subjects) 'Assign return value as an array
    set obSAS = Nothing
  end function
```

```
sub popSelect(selElement)
' Call the getsubjects function
  arrSub= getSubjects("C:\Documents and Settings","demo")
  set sel= document.getElementById("selSubs")
' Populate the select element with the value of subject from LST output
  for i =0 to ubound(arrSub)
    set objOption=document.createElement("OPTION")
    objOption.text= arrSub(i)
    objOption.value=arrSub(i)
    sel.add(objOption)
  next
end sub
' Use this function to append the list of subject to the sysparmVals
' variable in the runSAS subroutine
function listSelSub
  set ss=document.getElementById("selSubs")
  dim arrSubs()
  counter=0
  for i = 0 to (ss.options.length - 1)
    if (ss.options(i).selected) then
        redim preserve arrSubs(counter)
        arrSubs(counter)=ss.options(i).text
        counter=counter+1
    end if
  next
  listSelSub=join(arrSubs,",")
end function
```

### Add the SELECT element to the BODY tag

The onload event calls the popSelect subroutine to populate the options in the SELECT element. The SELECT element displays a drop down list displaying the subjects. The “Multiple” property is specified on the select element to allow more than one subject to be selected at a time.

```
<BODY onload="popSelect('selSubs')">

<!-- Add the select element after the radio button </fieldset> -->
<FIELDSET>
  <LEGEND>Select Subjects</LEGEND>
  <SELECT id="selSubs" size ="5" multiple></SELECT>
</FIELDSET>
```

## REFERENCING THE VALUE OF SYSPARM IN SAS

The value of sysparmVal contains the data set type (ADaM, Raw, or SDTM), a pipe character to separate the values, and a comma-delimited list of subjects. This is passed to the SAS environment through the SYSPARM command-line option. Within SAS, the value of the SYSPARM macro variable is referenced. Using the %SCAN function, the data set type and list of subjects are parsed from SYSPARM.

This example uses SYSPARM to provide all the information from the GUI to the SAS environment. If the GUI collects more than 200 characters, SYSPARM cannot be used . In this case, the HTA can write an external file containing the user input in a readable format and pass the location of the file to the SAS environment. The external file can then be read and parsed using an INFILE statement within a DATA step.

## CONCLUSION

Developing GUI applications to capture user input and pass that information to a SAS macro insulates the source code from the user. It allows the developer to design a user-friendly, intuitive interface that can expand the target user group beyond other programmers. HTAs can provide a SAS macro with the feel of an Microsoft Office application or webpage.

## REFERENCES

- SAS(R) 9.2 Integration Technologies: Windows Client Developer's Guide. Available At: http://support.sas.com/documentation/cdl/en/itechwcdg/61500/HTML/default/viewer.htm#titlepage.htm
- Introduction to HTML Applications (HTAs). Available at: http://msdn.microsoft.com/en-us/library/ms536496(v=vs.85).aspx
- FileDialog Object. Available at: http://msdn.microsoft.com/en-us/library/aa219843(v=office.11).aspx

## ACKNOWLEDGMENTS

I'd like to thank Norm Fox, Clinical Systems Analyst at Seattle Genetics, Inc, for his tutelage in HTML, CSS and JavaScript, and Steve Pearce, Associate Director of Statistical Programming at Seattle Genetics, Inc. for his support in writing this paper.

## CONTACT INFORMATION

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

Name: Shawn Hopkins
Enterprise: Seattle Genetics, Inc.
Address: 21823 30th Drive SE
City, State ZIP: Bothell, WA 98021
Work Phone: 425.527.2340
E-mail: Shopkins@seagen.com