Paper 228-30

Managing a SAS® Programming Environment using Data Access Pages and the Microsoft Office Data Source Control

Brian Fairfield-Carter, PRA International, Victoria, BC (Canada)
Stephen Hunt, PRA International, Victoria, BC (Canada)

ABSTRACT

A well-managed SAS programming environment offers convenient, centralized access to source code, authorship, completion and validation status, revision history, and validation documentation. It provides a quick view on progress, builds in effective process flow, enforces consistency, and eliminates redundancy and repetition. Toward this end, in programming for a clinical trial, we usually see the development of some sort of "Table of Programs" (TOP), containing information such as program name and location, author, completion date, validation status and date, revision history, and dependencies. In addition, either as part of the TOP or in a separate file, we also often see some sort of "Table of Tables, Figures and Listings (TFLs)", containing data such as titles and footnotes, TFL number, source data, analysis population, and output type.

This paper demonstrates the use of Microsoft Access as a data-base "back-end", in conjunction with a data-bound HTML "Data Access Page" (DAP) and the Microsoft Office Data Source Control (MSODSC) to integrate all of these components into a generalized, study-independent application. The DAP is created using tools provided in MS Access, and additional functionality, driven by user actions and MSODSC events, is built in by adding VBScript and JScript components to the DAP source code. A simple method for implementing version control using the "Concurrent Versions System" (CVS), by passing CVS shell commands to the windows command line interface, is also demonstrated.

INTRODUCTION

Although the functionality to be built into an application for managing a SAS programming environment is a matter likely to generate much debate, some simple underlying objectives can probably be established:

- "Study-independent" – the same interface can be used to manage different studies' data
- SAS program source code can be conveniently viewed and edited
- Completion and validation status, and programmer and reviewer comments, can be recorded and viewed
- Validation documentation can be created at the time the program is asserted to be "validated" (for example, when a 'Validated' radio button is checked in the interface)
- Table, Figure and Listing titles and footnotes, analysis populations, output 'areas' (interim, IND, ISS/ISE, CSR, etc.), and output file references can be entered centrally
- Search/find capability is supported
- Critical information (for example, program name, author, output file reference) can be 'locked' to prevent accidental alteration
- Access tables can be automatically exported as SAS datasets
- Some form of SAS program 'version control' and archiving can be supported

DATA ACCESS PAGES

Data Access Pages were first implemented in Microsoft Access 2000. A DAP is a 'data-bound' HTML page, meaning that it creates a dynamic connection to source data at start-up, and automatically flushes data to the database at specific 'events' (like clicking the 'save' or 'navigate' buttons). What makes DAPs an attractive development environment is that, while basic database operations (add delete, view, filter, navigate, etc.) are taken care of by default features, source code can be easily edited to modify and add functionality, meaning that any knowledge of Windows Scripting Technologies (VBScript, Jscript, dynamic HTML) can be taken advantage of.

CREATING A DATA ACCESS PAGE

The steps to create a Data Access Page are as follows:

1. First, open the database in Access (assuming that you have already created at least a shell database with at least one data table) and select the "Pages" tab
2. Next, double-click "Create data access page by using wizard", select the data table you want to create a page for, and select the fields to include
3. Save the page; double-click on the ".htm" file, and the page will open in Internet Explorer, displaying the first record from the source data table. Note the grey area below the data fields – this contains automatically-provided record navigation and edit controls ("new", "save", "delete", "next", "previous", etc.) for working with bound Access data.

HOW IT WORKS: "DATA-BINDING"

The DAP is said to be "data-bound" in that fields on the page are associated with fields on the source data table, and in that record navigation and edit controls (located in the gray region below the fields) change which record is selected on the source data table, and therefore the values displayed on the page. By scanning through the source code, it is possible to see how the database connection is established:

```
<a:ConnectionString>Provider=Microsoft.Jet.OLEDB.4.0;User ID=Admin;Data
Source=C:\db1.mdb;
```

A "connection string" specifies the "database engine" (Microsoft.Jet.OLEDB.4.0), and the target database is specified as the "Data Source". As will be seen later, the ability to change this connection string dynamically enables a single interface to connect to any number of databases, provided those databases conform to a consistent structure.

VBSCRIPT, JSCRIPT, AND DYNAMIC HTML

For SAS programmers looking for a starting point to expand their programming and applications development horizons, and for anyone wanting to throw together a dynamic application in a hurry, Windows Scripting Technologies probably provide the simplest and most accessible environment. For an introduction to SAS applications of VBScript and Jscript, in both the Windows Scripting Host and dynamic HTML environments, refer to Hunt *et al* (2005) and Fairfield-Carter *et al* (2005).

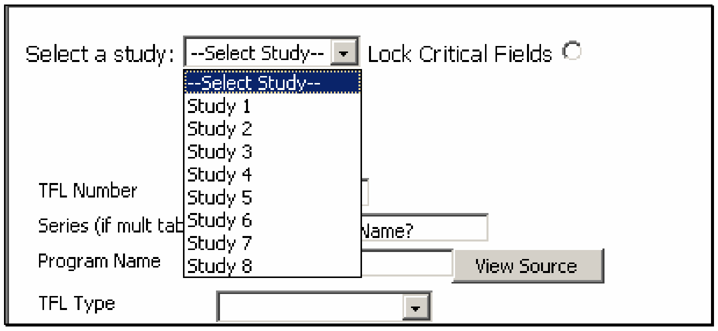
ADDING FUNCTIONALITY: SCRIPTING THE MICROSOFT OFFICE DATA SOURCE CONTROL

CONNECTING TO A STUDY DATABASE

Let's say we've established a standard (and simple) Access database structure for holding SAS programming information (program name, author, completion and validation status, titles and footnotes, etc.) in a number of different trials. In order to implement a generic DAP, we need some way of allowing it to connect to any of a number of databases. In order to do this, the first thing we want to do is remove the hard-coded database reference built into the page:

<a:ConnectionString>Provider=Microsoft.Jet.OLEDB.4.0;User ID=Admin;Data
Source=~~C:\db1.mdb~~;

By setting "Source=", the page launches without connecting to a database, so now we need some way of supplying the missing piece of the connection string. One way of doing this is to create a drop-down list of study names, and attach an "OnChange" event script to it that builds and submits an appropriate connection string.



The "--Select Study--" drop-down list is implemented as a simple selection list:

```
<SELECT id=StudySelect></SELECT>
```

The list is populated via a start-up script when the page loads:

```
<SCRIPT LANGUAGE="JScript">
 // Function to populate the study selection list
 function GetItemTest(vbarray, Elements_)
 {
   var i;
   var SArray = new VBArray(vbarray);
   for (i = 1; i <= Elements_; i++)
   {
          var oOption = document.createElement("OPTION");
          StudySelect.options.add(oOption);
          oOption.innerText = (SArray.getItem(1, i));
          oOption.Value = (SArray.getItem(1, i));
    }
 }
</SCRIPT>
<SCRIPT LANGUAGE="VBScript">
'Create VB array to hold study names, and call function to populate selection list
  Dim StudyArray(1,9)
      StudyArray(1,1)="--Select Study--"
      StudyArray(1,2)="Study 1"
      StudyArray(1,3)="Study 2"
      ...(etc.)...
  GetItemTest StudyArray, UBound(StudyArray,2)
</SCRIPT>
```

An “OnChange” event script is then added to respond to the user’s selection. This could be simplified to pass a hard-coded database reference for each item in the selection list, but a slightly more artful method uses the page’s URL (assuming that the page is in the same root directory as the target databases) to retrieve the root directory, and then concatenate the rest of the path based on standard directory structure conventions.

```
  <SCRIPT For="StudySelect" Event="OnChange" LANGUAGE="VBScript">
  'To allow path-independence and study-independence --> use the root directory
  ' and standard directory structure 'to 'ReLoad' the correct Access database
   Dim iSelectedIndex
   iSelectedIndex = StudySelect.SelectedIndex
   document.Title=StudySelect.options(iSelectedIndex).text & " TOP Entry Page"
   ReconnectDataAccessPage Left(GetRoot,14) & StudySelect.options(iSelectedIndex).text
      & "\Data\Environment\TFL Specs.mdb"
  </SCRIPT>
```

The “GetRoot” and “ReconnectDataAccessPage” functions are defined in a start-up script:

```
  <SCRIPT LANGUAGE="VBScript">
    Const intro = "Data Source="
    Function ReconnectDataAccessPage(todb)
      str = MSODSC.ConnectionString
      n = InStr(1, str, intro) + Len(intro) - 1
      connect1 = Left(str, n)
      connect2 = Right(str, Len(str) - n)
      n = InStr(1, connect2, ";")
      pathname = Left(connect2, n - 1)
      connect2 = Right(connect2, Len(connect2) - n + 1)
      newstr = connect1 & todb & connect2
      MSODSC.ConnectionString = newstr
    End Function
    Function GetRoot
      DBase=split(document.URL,")",-1,1) 'Start with the full URL
      GetRoot=Replace(GetRoot,"file://","")  'Get rid of prefix
    End Function
  </SCRIPT>
```

Selecting an entry in the drop-down list will now pass a database reference, created through concatenation as “\\server\<root>\<selected study>\Data\Environment\TFL Specs.mdb”, where <root> is returned by the “GetRoot” function, to the “ReconnectDataAccessPage” function, which pulls apart the MSODSC.ConnectionString and inserts the database reference.

LOCKING ‘CRITICAL’ FIELDS

Once you’ve created an entry for a program, there are certain pieces of information that you probably want to avoid accidentally changing. These might include things like program name, author, references to output files, analysis population, output type, and completion status. One way to implement this would be to add a “lock critical fields” radio button to the interface, so that the user had to deliberately unlock the record before any modifications could be made (and, for safety, have the lock automatically turned back on each time a new record was loaded):

```
  Lock Critical Fields<input type="Radio" name="LockFields">
```

The implementation shown here uses a set of global variables (initialized in a start-up script so that they’re “in scope” in all event scripts) to capture the values passed from the database at the time the record loads, and to simply over-write anything entered in the interface with these values when the record is released, if the “lock critical fields” flag has not been turned off.

```
  <SCRIPT language=vbscript>
```

```
    'Declare globals -- used to hold "critical" values when record loads
    '--------------------------------------------------------------------
    Dim HoldTFLNUMValue , HoldSERIESValue , HoldPROGNAMEValue , HoldPGMEDValue
    Dim HoldPGMDATEValue , HoldList1Value , HoldList2Value
    Dim HoldList3Value , HoldList4Value
    Dim LockFieldsStartValue   'Global to hold the 'lock fields' radio button value
    Dim NumberOfRecordsAtStart 'Number of records in the connected dataset at startup
  </SCRIPT>
```

Nine fields were selected as being ‘critical’: TFL output reference number (consisting of table/figure/listing number plus series number), program name, completion status (programmed yes/no and date), output type (table, figure, listing, derived dataset), analysis population, area (interim analysis, final report, etc.), and programmer name. In addition to declaring globals to hold these values, two more globals are declared: the “LockFieldsStartValue” holds the ‘current’ status of the ‘lock fields radio button (this is necessary because clicking this button reverses its value, and you need some way of determining what the original value was), and “NumberOfRecordsAtStart” holds the number of records on the bound Access table when the DAP first loads (this allows new records to be added without triggering ‘locked fields’ warnings).

```
  <SCRIPT for=document event=onclick language=vbscript>
   'Set global to be used in event script (capture value BEFORE the radio
   ' button reverts in response to user action)
     LockFieldsStartValue=LockFields.Checked
  </SCRIPT>
```

Since the “OnClick” event of the HTML document fires before the “OnClick” event of the ‘lock fields’ radio button, the ‘current’ value of the radio button can be captured each time the user clicks in the interface. If the user happened to have clicked on the radio button, the event can be handled correctly by making a comparison between the ‘starting’ value, and the LockFields.Checked value resulting from the event:

```
  <SCRIPT FOR="LockFields" EVENT="OnClick" language=vbscript>
    If LockFieldsStartValue=True Then
      LockFields.Checked=False
    Else
      LockFields.Checked=True
    End If
    If LockFields.Checked=False Then
      MsgBox("!!WARNING -- YOU HAVE UNLOCKED CRITICAL FIELDS IN THIS PAGE")
    End If
  </SCRIPT>
```

The behavior of the ‘lock fields’ radio button has to be scripted explicitly, as the default behavior only allows the button to go from ‘unchecked’ to ‘checked’, but not the reverse. A message box has been added to warn the user when critical fields are exposed to alteration.

The following script uses the “DataPageComplete” event of the MSODSC to capture the number of records on the bound Access table at the time the page loads, so that existing records can be distinguished from new records, and to set the ‘lock fields’ button to ‘checked’ (fields should be automatically locked when the page first loads, as well as each time a record is loaded).

```
  <SCRIPT for=MSODSC event=DataPageComplete(dscEventInfo) language=vbscript>
    NumberOfRecordsAtStart=MSODSC.DataPages(0).Recordset.RecordCount
     'Set the Critical Fields Lock flag to "On" at the time the data page loads
     LockFields.Checked=True
     LockFieldsStartValue=True
  </SCRIPT>
```

The following script uses the “Current” event of the MSODSC, which fires each time a record loads, to, again, lock critical fields, and also to capture the initial values of all ‘critical’ fields. If changes are subsequently made to values displayed in the interface without the critical fields being unlocked, these held values will over-write the new values and ultimately be flushed back to the database.

```
  <SCRIPT for=MSODSC event=Current(dscEventInfo) language=vbscript>
    'Set the Critical Fields Lock flag to "On" at the time the record loads
     LockFields.Checked=True
     LockFieldsStartValue=True
    'Capture values on all critical vars at the time the record loads
    '-----------------------------------------------------------------
    If MSODSC.DataPages(0).Recordset.Bookmark <= NumberOfRecordsAtStart Then
```

```
        HoldTFLNUMValue = MSODSC.DataPages(0).Recordset.Fields(1).Value
        HoldSERIESValue = MSODSC.DataPages(0).Recordset.Fields(2).Value
        HoldPROGNAMEValue = MSODSC.DataPages(0).Recordset.Fields(3).Value
        HoldPGMEDValue = MSODSC.DataPages(0).Recordset.Fields(4).Value
        HoldPGMDATEValue = MSODSC.DataPages(0).Recordset.Fields(5).Value
        HoldList1Value = MSODSC.DataPages(0).Recordset.Fields(18).Value 'TFLTYPE
        HoldList2Value = MSODSC.DataPages(0).Recordset.Fields(19).Value 'POP
        HoldList3Value = MSODSC.DataPages(0).Recordset.Fields(20).Value 'PGMER1
        HoldList4Value = MSODSC.DataPages(0).Recordset.Fields(22).Value 'AREAB
      End If
    </SCRIPT>
```

The following script uses the "BeforeUpdate" event of the MSODSC to over-write any altered values in critical fields with values captured when the record was loaded, before data are flushed back to the database. This over-writing is done only if the 'lock fields' button is checked, and the record has not just been added (as determined by the comparison between "NumberOfRecordsAtStart" and the bookmark number of the current record).

```
    <SCRIPT for=MSODSC event=BeforeUpdate(dscEventInfo) language=vbscript>
     If LockFields.Checked=True and MSODSC.DataPages(0).Recordset.Bookmark <=
    NumberOfRecordsAtStart Then
         MSODSC.DataPages(0).Recordset.Fields(1).Value = HoldTFLNUMValue
         MSODSC.DataPages(0).Recordset.Fields(2).Value = HoldSERIESValue
         ...(etc.)...
       MsgBox("!!CRITICAL FIELD LOCK IS ON -- CRITICAL FIELDS HAVE NOT BEEN UPDATED")
     End If
    </SCRIPT>
```

In order to tie up a few loose ends, protection is also implemented against accidental deletion of records:

```
    <SCRIPT for=MSODSC event=BeforeDelete(dscEventInfo) language=vbscript>
       If LockFields.Checked=True Then
          MsgBox("!!CRITICAL FIELD LOCK IS ON -- CAN NOT DELETE RECORD")
          dscEventInfo.returnValue = False
       End If
    </SCRIPT>
```

After a record has been deleted, the number of records in the database will be one fewer, so the "NumberOfRecordsAtStart" variable must be decremented:

```
    <SCRIPT for=MSODSC event=AfterDelete(dscEventInfo) language=vbscript>
        NumberOfRecordsAtStart=NumberOfRecordsAtStart-1
    </SCRIPT>
```

As with any applications development environment, you will run into a certain amount of bizarre and counter-intuitive behavior with DAPs. With the MSODSC, the first record displayed when a record set is loaded seems to get a "BeforeUpdate" event, even if no changes have been made to any of its values. As a result, if you set a filter toggle (for example, to only display records where TFL type is "Table") in one database, and then load a new database, you can end up with held values over-writing correct values. It is therefore a good idea to explicitly set held values to null on each MSODSC "RecordExit" event.

```
    <SCRIPT for=MSODSC event=RecordExit(dscEventInfo) language=vbscript>
      HoldTFLNUMValue=""
      HoldSERIESValue=""
      ...(etc.)...
    </SCRIPT>
```

IMPLEMENTING SEARCH/FIND FUNCTIONALITY

Once you have more than a dozen or so records, you'll need some convenient way of navigating to specific records. There are a number of different ways of getting search criteria from the user (two of which are shown here), and once these criteria are captured the process of locating the target record follows a fairly similar pattern: first, the access table is 'cloned', and this clone is queried to find the target record; this record is then 'book-marked', and the current 'book-mark' in the record set displayed in the access page is set to the same value, so that that record is displayed in the interface.

METHOD 1

The first method of retrieving search criteria is the simplest. It consists of a simple InputBox, displayed through an event script for a "Find" button, into which the user can type a search expression:

```
    <INPUT type=button value="Find" name=Find_>
```

The event script for this button pops up the input box:

```
<SCRIPT language=vbscript event=onclick for=Find_>
FindExpression = InputBox("Enter a valid search expression." & Chr(13) & Chr(13) &
"[Example: PROGNAME=" & Chr(34) & "my_program" & Chr(34) & "]")
```

The search string is then ‘parsed’ in order to determine 2 ingredients: first, what field number on the access table should be searched, and second, what value of that field should be found:

```
'Parse the expression to pass parameters to generic search string
   FindValue= Right(FindExpression,Len(FindExpression)-Instr(FindExpression,"="))
     If Instr(FindValue,Chr(34)) Then
       FindValue=CStr(Replace(FindValue,Chr(34),""))
     End If
   Select Case UCASE(Left(FindExpression,Instr(FindExpression,"=")-1))
          Case "TFLNUM"   FieldNumber=1
          Case "SERIES"   FieldNumber=2
          Case "PROGNAME" FieldNumber=3
          Case "PGMED"    FieldNumber=4
          Case "TFLTYPE"  FieldNumber=18
          Case "POP"      FieldNumber=19
          Case "PGMER1"   FieldNumber=20
          Case "AREAB"    FieldNumber=22
          Case Else       FieldNumber=9999  '(error code)
   End Select
```

Now the search can be conducted (on the cloned record set). Note that to save space, no error handling is included here:

```
 'Clone the recordset.
   Dim rs
   Set rs = MSODSC.DataPages(0).Recordset.Clone
 'Execute the search
    On error resume next
    rs.MoveFirst
     If rs.Fields(FieldNumber).Value=FindValue Then
        MsgBox("You are already on the record you're looking for.")
     Else
      End_=0
      Do Until End_=1
       rs.MoveNext
       If rs.Fields(FieldNumber).Value=FindValue or (rs.bof) or (rs.eof) Then End_=1
      Loop
     End If
   End If
```

The user is then informed if the search on the cloned record set produces nothing; otherwise, the bookmark on the cloned record set is set as the bookmark on the original record set, so that the target record is displayed in the interface.

```
  'Check search results for success.
   If (rs.bof) or (rs.eof) Then
       Msgbox "No Record found matching those criteria",,"Search Done"
       Exit Sub
   End If
   If FindExpression>"" and FieldNumber<9999 Then
     MSODSC.DataPages(0).Recordset.Bookmark = rs.Bookmark
   End If
```

METHOD 2

The second method exploits the “OnClick” event of the HTML document to write the name of the last ‘searchable’ element the user selected to a global:

```
<SCRIPT language=vbscript>
  Dim srcElementFind '(Global)
</SCRIPT>
<SCRIPT for=document event=onclick language=vbscript>
If UCase(window.event.srcElement.ID)="TFLNUM" or
UCase(window.event.srcElement.ID)="SERIES" or
 ...(etc.)...
Then
   srcElementFind = window.event.srcElement.ID
End If
</SCRIPT>
```

If the user then clicks inside the “Find” text area on the DAP, an event script is fired which queries the Access table for all possible values of the last searchable field selected, and writes these values into a temporary HTML file, which is then displayed as a modal dialog (in other words, this dialog holds the focus until it is closed). The user can then select one of the displayed values, which is then captured as a document ‘cookie’, and the dialog is automatically closed. This ‘cookie’ is then concatenated to the selected field name, and displayed in the text area as a search string.

First, the text field and ‘find’ button are created:

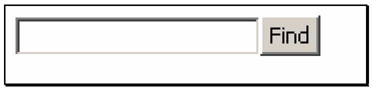
```
   <INPUT TYPE="Text" NAME="ProgNameFind">
   <INPUT TYPE="Button" NAME="Find1" VALUE="Find">
```

An “OnClick” event script is then added for the text field:

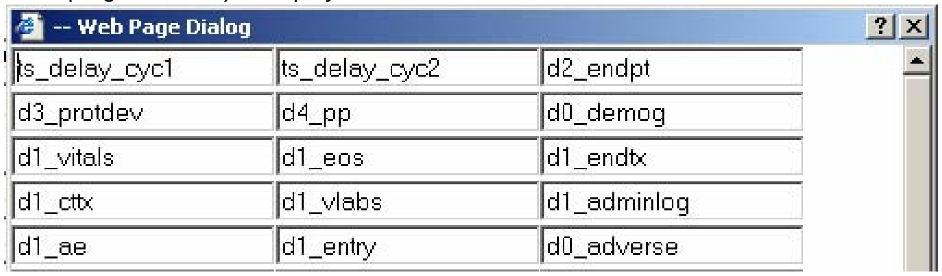
```
   <SCRIPT FOR="ProgNameFind" EVENT=OnClick LANGUAGE="VBScript">
       If srcElementFind>"" Then
      Select Case UCase(srcElementFind)
             Case "TFLNUM"    FieldNumber=1
             Case "SERIES"    FieldNumber=2
             Case "PROGNAME"  FieldNumber=3
             Case "PGMED"     FieldNumber=4
             Case "DROPDOWNLIST1" FieldNumber=18
             Case "DROPDOWNLIST2" FieldNumber=19
             Case "DROPDOWNLIST3" FieldNumber=20
             Case "DROPDOWNLIST4" FieldNumber=22
             Case Else FieldNumber=9999
      End Select
     'Write an HTML selection page as a simple text document
      Dim htmfso, htmdoc
      Set htmfso=CreateObject("Scripting.FileSystemObject")
      Set htmdoc=htmfso.CreateTextFile(Left(GetRoot,Instr(GetRoot,"\SAS")+4) &
   "select.htm",2)
      Dim rset_
      Set rset_ = MSODSC.DataPages(0).Recordset.Clone
      rset_.MoveFirst
      End_=0
      i_=0
       Do Until End_=1
         If i_=0 Then
     'Embed an "OnClick" event script
   htmdoc.WriteLine("")
   htmdoc.Write("<SCR" & "IPT for=document event=OnClick language=vbscript>" & Chr(13))
   htmdoc.Write("If window.event.srcElement.ID>"& Chr(34) & Chr(34) & " Then"& Chr(13))
   htmdoc.Write("Document.Cookie = window.event.srcElement.Value" & Chr(13))
   htmdoc.Write("window.close" & Chr(13))  'To automatically close the dialog
   htmdoc.Write("End If" & Chr(13))
   htmdoc.Write("</SCR" & "IPT>" & Chr(13))
         End If
         i_=i_+1
   'Create a text field to hold each value from the record set
    htmdoc.Write("<INPUT TYPE=" & Chr(34) & "Text" & Chr(34) & " ID=" & Chr(34) &
   srcElementFind & Chr(34) & " VALUE=" & Chr(34) & rset_.Fields(FieldNumber).Value &
   Chr(34) & ">")
        rset_.MoveNext
        If (rset_.bof) or (rset_.eof) Then End_=1
     Loop
     htmdoc.close
     'Clear cookie
      Document.Cookie = ""
     'Pop up the HTML selection page created above as a modal dialog
      dim fResync
      fResync = showModalDialog("select.htm", "dialogHeight: 450px; dialogWidth: 600px;
   center: yes; resizeable: no; status: no;")
```

```
     'Display the "Find" string in the text box
      ProgNameFind.Value = srcElementFind & "=" & Chr(34) & Document.Cookie & Chr(34)
    Else
      Find_.OnClick 'Otherwise, submit an onclick event for the other "Find" button
    End If
  </SCRIPT>
```

The following sequence of events can now occur in the user interface:
1. The user clicks in the 'find' text area:

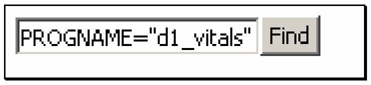


2. If no searchable field has previously been selected, the dialog box for the first search method is displayed (by firing a "Find_.OnClick" event); otherwise, a modal dialog containing all values of the last searchable field selected (in this case, program name) is displayed:



ts_delay_cyc1	ts_delay_cyc2	d2_endpt
d3_protdev	d4_pp	d0_demog
d1_vitals	d1_eos	d1_endtx
d1_cttx	d1_vlabs	d1_adminlog
d1_ae	d1_entry	d0_adverse

3. When an entry is selected, the modal dialog is closed, and the selection is concatenated to the field name to create a search string:



PROGNAME="d1_vitals"

When the user then clicks on the "Find" button, an event script (identical to the one shown above) is fired that executes the search.

VIEWING SAS PROGRAMS AND CREATING PROGRAM TEMPLATES

While updates are being made in the DAP, it's convenient to be able to view and edit the SAS source code without having to navigate through Windows Explorer. One way of doing this is simply to add a button to the interface that takes the program name on the currently displayed record, and opens the program in the SAS editor. This process also provides an opportunity to generate a 'shell' program for entries that don't yet have source code associated with them. In order to do this, the event script for the button must build the full path and file name for the target program as a text string, based on the root directory, selected study, and standard directory structure.

```
  <INPUT type=button value="View Source" name=Source_>
```

The 'OnClick' event script for this button is as follows:

```
  <SCRIPT FOR="Source_" EVENT="OnClick" LANGUAGE="VBScript">
  'Get the index number for the selected entry on the StudySelect option list
    Dim iSelectedIndex
    iSelectedIndex = StudySelect.SelectedIndex
```

The path name of the target directory is constructed as a text string, by concatenating the root directory to the study-level directory (the selected entry from the StudySelect list) and to the correct development-area directory:

```
  'For TFLs:
  If UCase(MSODSC.DataPages(0).Recordset.Fields(18).Value)="TABLE" or
  UCase(MSODSC.DataPages(0).Recordset.Fields(18).Value)="FIGURE" or
  UCase(MSODSC.DataPages(0).Recordset.Fields(18).Value)="LISTING" Then
  ProgPath=Left(GetRoot,14) & StudySelect.options(iSelectedIndex).text &
  "\SAS\DEV\REPORT"
   '...(etc. - further sub-setting by analysis: Interim, IND, etc.)...
  End If
  'For derived datasets:
  If UCase(MSODSC.DataPages(0).Recordset.Fields(22).Value)="DERIVE" Then
  ProgPath=Left(GetRoot,14) & StudySelect.options(iSelectedIndex).text &
  "\SAS\DEV\DERIVE"
   '...(etc. - further sub-setting by derived dataset type)...
  End If
```

If the program doesn't yet exist, it is created as a 'shell' program (consisting of the standard program header):

```
    Dim objFS, ProgText
    Set objFS = CreateObject("Scripting.FileSystemObject")
    If objFS.FileExists(ProgPath & "\" & PROGNAME.Value & ".sas")=False Then
      Set ProgText=objFS.CreateTextFile(ProgPath & "\" & PROGNAME.Value & ".sas",2)
      With ProgText
   .WriteLine("/*______________________________________________________" & Chr(13))
   .WriteLine("STUDY : PROTOCOL " & StudySelect.options(iSelectedIndex).text & Chr(13))
   .WriteLine("PROGRAM NAME : " & PROGNAME.Value & ".sas" & Chr(13))
   .WriteLine("PROGRAM TYPE : SAS 8.2" & Chr(13))
   .WriteLine("PURPOSE : " & Chr(13))
   .WriteLine("PROGRAMMER : " & MSODSC.DataPages(0).Recordset.Fields(20).Value &
   Chr(13))
   .WriteLine("DATE : " & Date() & Chr(13))
   ...(etc.)...
      End With
      ProgText.Close
      End If
```

A SAS session is launched, and either the newly-created shell program or the existing program is opened:

```
     Dim OSAS
     Set OSAS = CreateObject("SAS.Application.8")
     OSAS.Visible=True
     CommandString="inc '" & ProgPath & "\" & PROGNAME.Value & ".sas'"
     OSAS.Command(CommandString)
   </SCRIPT>
```

Note that when generating the shell SAS program as a text stream, concatenating the ASCII character 13 (" & Chr(13)") produces a line break; an alternative is to insert a ".Write("")" line.

CREATING QC DOCUMENTS

Programmers have a common tendency to procrastinate on the creation of QC documentation. This makes the addition of this functionality to a managed programming environment desirable, to facilitate an efficient and logical work-flow. With the functionality shown so far, we can imagine the following work-flow steps:

1. Programmer creates a new entry in the DAP
2. Programmer clicks on the 'view source' button to create and open a new shell program
3. Programmer completes and tests the new program
4. Programmer checks the 'programmed' radio button and supplies a date
5. QC programmer navigates to the record for this new program
6. QC programmer clicks 'view source', reviews the code, runs the program, and reviews the log
7. QC programmer clicks the 'tested' radio button and supplies a date

Now the QC programmer is required to fill out a QC checklist to document validation activities. The process of creating a correctly-named copy of the checklist template, in the correct directory and referencing the correct program and output, can be attached to the action of checking the 'tested' radio button via an "OnClick" event script:

```
   ' Create an instance of Word
   Dim objwd
   Set objwd = CreateObject("Word.Application")
   objwd.visible=false
   ' Create a blank document and insert checklist template
   objwd.documents.add
   objwd.Selection.InsertFile("\\server\<directory>\QC Checklist.doc")
   ' Replace program name place-holder with program name
   objwd.selection.wholestory
   oldtext1="[Program Name]"
   objwd.Selection.find.execute oldtext1
   objwd.selection.typetext MSODSC.DataPages(0).Recordset.Fields(3).Value
   ' Replace program author name place-holder with author name
   ...(etc. – code reviewer, date, output reference)...
```

The checklist is now saved in a QC directory mirroring the programming directory structure (the full path and file name are created through concatenation using methods already illustrated):

```
   ' Save the document and exit Word
   objwd.activedocument.saveas("\\server\<directory>\program_name.doc")
   objwd.ActiveDocument.Close()
   objwd.Application.Quit()
```

ARCHIVING AND MOVING TO PRODUCTION

Once a SAS program is validated, it can be archived and moved into production. This is done by retaining a copy (with the current system date appended to the file name) in the development area, and copying the program to the production area:

```
   Dim fs
   Set fs = CreateObject("Scripting.FileSystemObject")
   Dev_="<DEV path>\" & PROGNAME.Value & ".sas"
   Archive_="<DEV path>\" & PROGNAME.Value & "_" & date() & ".sas"
   Prod_= Replace(UCase(<DEV path>),"\SAS\DEV","\SAS\PROD") & PROGNAME.Value & ".sas"
   fs.CopyFile Dev_, Archive_
   fs.CopyFile Dev_, Prod_
```

EXPORTING ACCESS DATA TO SAS VIA THE ACTIVEX DATA OBJECT

A final, vital component of this managed SAS programming environment is the ability to export Access data to SAS. This is necessary as some of the Access data (i.e. table titles and footnotes) are used directly in producing SAS output, and other data (i.e. analysis population, validation status) are used indirectly to control production runs. Rather than relying on PROC ACCESS or 'dynamic data exchange', we chose to export SAS data directly from the DAP through the ActiveX Data Object, by means of an "OnClick" event script attached to a "To SAS" button. Because the event script is virtually identical to the Access-->SAS Windows Scripting Host script provided by Hunt, *et al* (2005), it is not reproduced here.

CONCLUSION

Data Access Pages and the Microsoft Office Data Source Control provide a simple but powerful collection of tools for creating an application to manage a SAS programming environment. By automating repetitive tasks, providing centralized access to source code and completion/validation status, and exploiting the full capability of dynamic HTML, a logical work-flow can be constructed where, for example, lead programmers have ready access to authorship and completion/validation status, and code reviewers have both a convenient way of determining when QC checks should be performed, and a painless way of producing validation documentation.

FUTURE DEVELOPMENTS: INTEGRATION WITH CVS (CONCURRENT VERSIONS SYSTEM) FOR SOURCE CONTROL

With the growing interest in maintaining a revision history for SAS analysis programs, a good next step for a DAP-based system would be the integration of some sort of source control application. The "Concurrent Versions System" (CVS), because it is an open-source GPL application, is perfectly suited to such integration (see Williams, 2004). As an initial step, CVS commands can be passed very easily by an event script to the windows command prompt. For example, a "CVS Update" could be called from the "view source" event script so that the current version of the program would be downloaded from the CVS repository prior to being opened:

```
   Dim WshShell_
   Set WshShell_ = CreateObject("WScript.Shell")
   WshShell_.Run "cmd /K cd <working directory> & cvs -d <working directory> checkout
   <module name>"
```

Similarly, a CVS Commit could be added to the "tested" event script, to commit a version to the repository once it had been validated:

```
   WshShell.Run "cmd /K cd <working directory> & cvs -q commit -m " & Chr(34) & Chr(34)
   & " program.sas"
```

REFERENCES

- Fairfield-Carter, Brian, Sherman, Tracy, and Hunt, Stephen (2005), "Instant SAS® Applications with VBScript, Jscript, and dHTML", Proceedings of the 30th Annual SAS Users Group International Conference.
- Hunt, Stephen, Sherman, Tracy, and Fairfield-Carter, Brian (2005), 'An Introduction to SAS® Applications of the Windows Scripting Host', Proceedings of the 30th Annual SAS Users Group International Conference.
- Williams, Tim (2004), 'Version Control on the Cheap: A User-Friendly, Cost-Effective Revision Control System for SAS®', Proceedings of the 2004 Pharmaceutical Industry SAS Users Group Conference.

CONTACT INFORMATION

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

Brian Fairfield-Carter, Stephen Hunt
PRA International, 600-730 View Street, Victoria, BC, Canada V8W 3Y7
Email: FairfieldCarterBrian@PRAIntl.com, HuntStephen@PRAIntl.com