Integration of Microsoft® Office and SAS® Software: Implementation of a Graphical User Interface and Automation

Rubin Nan, PRA International, Lenexa, KS
David Mullins, PRA International, Lenexa, KS

ABSTRACT
This paper describes techniques that use MS-Office embedded Visual Basic tools or the Visual Basic software to create a Graphical User Interface (GUI). Using this interface, users can direct the SAS® session, DATA steps, or SAS procedures to run and even input SAS macro parameters (macro variable values) from a front end to resolve macro variables in SAS programs.

To clarify our description, we use simple examples to demonstrate how to create a GUI, to set up component properties, and to write the Visual Basic and the SAS code to implement the integration and automation.

INTRODUCTION
SAS-related tasks need experienced SAS programmers to write the initial SAS code. However, some tasks such as routine data analysis, database maintenance, figure or table report generation, and documentation based on an existing standard database structure may not need employees to learn or know SAS.

All these tasks are better done by an automation process through GUI implementation; and the advantages of using GUIs have been well-elaborated by McGowan (McGowan, 2002).

Since MS-Word, MS-Excel, and MS-PowerPoint are all embedded with Visual Basic tools, users can integrate these tools with SAS to develop their own systems for their various tasks. Additionally, since the Visual Basic software can generate executable files (.EXE file), users can also integrate Visual Basic software with SAS to develop independently executable software.

Besides the aforementioned tasks that can be done in a GUI environment, clients’ needs can be met through an Intranet or the Internet by integrating SAS with either the Visual Basic software or MS-FrontPage embedded Visual Basic tools. This allows clients to share project information such as deliverables, documents, databases, etc.

Selecting MS-Visual Basic or MS-Office embedded Visual Basic tools for designing the GUI platform has two obvious advantages:

- MS-Office is used globally in business.
- The ODBC techniques of data exchange between SAS and MS-Office (e.g. MS-Excel and MS-Access) are well-developed.

Through the integration, all tasks such as data manipulation and statistical analysis can be done from a front end of MS-Office, and all SAS-generated output can be directly exported as various MS format files for either storage or as deliverables to clients.

This paper describes the techniques of GUI implementation by using Visual Basic or the MS-Office embedded Visual Basic tools to drive the SAS session in order to complete various tasks.

The retrieval of output into a GUI after SAS execution involves relatively complicated Visual Basic skills, and will be beyond the scope of this paper. Also, the techniques described in this paper are based on our experiences and subject to further development.

DEVELOPMENT OF A GUI AND AUTOMATION

Visual Basic code can submit SAS code to a SAS session. This can be done with the Visual Basic script language and one command button, without the need to write a separate SAS program for each task of opening a SAS session, running an existing SAS program, assigning a value to a macro variable, etc. (see the following Parts 1, 2, and 3). This coding is simple and has little chance to invite errors.
However, it is not good practice to use the Visual Basic script language to do a complicated SAS task, because the SAS program is embedded in the Visual Basic code and is not portable. This increases programming complexity and difficulties in error checking during the SAS and Visual Basic program developmental phase. Therefore, this task is better done by the following:

- A relatively complicated GUI consisting of command buttons, text boxes, list boxes, or options, etc., through which users can input parameters (macro variable values).
- Visual Basic script language that integrates the Visual Basic and SAS tasks.
- A separate SAS program or macro (see the following Part 4).

In the following we describe the development of automation with a GUI for the various tasks to be done:

1. **Drive the SAS session to open**: There are two ways to drive the SAS session to run from a GUI designed by Visual Basic - using the shell function and OLE. Both of these ways just need a command button.

   Assume a command button was set up with a property name: Run SAS, and SAS.EXE is located at C:\Program Files\SAS\sas.exe. The code in the Visual Basic module to use the shell function will be:

   ```vba
   Private Sub Run_SAS_Click()
   Rc = shell("C:\Program Files\SAS\sas.exe")
   End Sub
   ```

   While the code in the Visual Basic module to use OLE will be:

   ```vba
   Private Sub Run_SAS_Click()
   Dim olesas As Object
   Set olesas = CreateObject("SAS.Application")
   End Sub
   ```

2. **Drive the SAS program to run**: Once the SAS session is open, the program can be submitted from the session. However, this event can be done with the GUI by adding one line of SAS code to the Visual Basic module.

   Assume a SAS program, autoexec.sas, needs to be run. The code in the Visual Basic module to use the shell function will be:

   ```vba
   Private Sub Run_sas_Click()
   Rc = shell("C:\Program Files\SAS\sas.exe -AUTOEXEC C:\SAS\autoexec.SAS")
   End Sub
   ```

   While the code in the Visual Basic module to use OLE will be:

   ```vba
   Private Sub Run_sas_Click()
   Dim olesas As Object
   Set olesas = CreateObject("SAS.Application")
   olesas.Submit("data; %inc 'C:\SAS\autoexec.SAS'; run;")
   End Sub
   ```

   Please note that one shell function can submit only one existing SAS program or SAS session to run; whereas OLE can use multiple submit function statements to run multiple SAS programs, data steps, procedures, or macros.

3. **Drive SAS statements, DATA steps, or procedures to run**: The Visual Basic code can submit SAS code to the SAS session. The following is an example of using the PRINT procedure to print a dataset located at C:\SAS.

   ```vba
   Private Sub Run_sas_Click()
   Dim olesas As Object
   Set olesas = CreateObject("SAS.Application")
   olesas.Submit("libname dm 'C:\SAS'; proc print data=dm.demo; run;")
   End Sub
   ```

4. **Drive macros or programs with macro variables to run**: Complicated tasks such as driving macros or programs with macro variables to run need the following:

   - A GUI with the various Visual Basic components.
   - Visual Basic code.
• A separate SAS program or macros that is both independent of the Visual Basic code. This allows programmers to develop the SAS program independently of the GUI.

To clarify, we use an example of PROC CONTENTS to get data definition information such as variable name, format, length, and type; and then save the output into a spreadsheet as a report or document file.

The dataset could be from a different project, which means it may be located in a different folder. Each project may have its own autoexec.sas file to define its data libraries, macros, page size, line size, font, etc.

To do such a task, we need to design a GUI (shown as Figure 1) with various components, to set up the component’s properties (shown in Table 1), to write the Visual Basic code (shown as attached Visual Basic code) and the SAS code (shown as attached SAS© program, print.sas).

The GUI components includes a common button to drive the SAS session to run, a text box and two list boxes (text boxes are also doable) to allow users to respectively input 1) the folder where the autoexec.sas, print.sas, and the .XLS document file are located, 2) the data library name, and 3) the dataset name.

The SAS code can be a SAS program with macro variables or just a SAS macro. Although we use one text box and two list boxes as examples to input parameters (macro variable values), they can be replaced by each other or set up for the possibility that a text box needs to contain multiple, predetermined parameter values; or if a list box is not necessary to contain multiple, predetermined parameter values.

Figure 1 - GUI design example to run the print.sas program, located at C:sas, to print the dataset name, variable length, type, format, and label as a file into MS-Excel; using one or all dataset(s) located at the data library, Master or Analysis.

Table 1 - GUI component’s property setup

<table>
<thead>
<tr>
<th>Label</th>
<th>Component</th>
<th>Component Name</th>
<th>Default value</th>
<th>Caption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Folder</td>
<td>Text Box</td>
<td>Txtfolder</td>
<td>C:sas</td>
<td></td>
</tr>
<tr>
<td>Data Library Name</td>
<td>List Box</td>
<td>Lstlib</td>
<td>Master, Analysis</td>
<td></td>
</tr>
<tr>
<td>Dataset to Print</td>
<td>List Box</td>
<td>Lstdsn</td>
<td>Demo, <em>All</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Command Button</td>
<td>Run SAS</td>
<td></td>
<td>Run SAS</td>
</tr>
</tbody>
</table>

5. Automation control: The script language for controlling the SAS session from the GUI is available in Visual Basic. The script includes checking whether the SAS session is busy, toggling the SAS session between visible and invisible, setting the main SAS window title of the SAS session, ending the SAS session, checking errors, etc. The attached example Visual Basic code shows just part of the automation control. Others can be seen in detail in Reference #2.

Following is the attached Visual Basic code:

Private Sub Run_SAS_Click()
    'Error checking starts
    On Error GoTo Err_Run_sas_Click
    Dim olesas As Object
    Dim txtfolder As String
    Dim lstlib As String
    Dim lstdsn As String
folder = txtfolder.Value
lib = lstlib.Value
dsn = lstdsn.Value

Set olesas = CreateObject("SAS.Application")
olesas.Submit ("data;
    %let dsn=" & dsn & ";
    %let lib=" & lib & ";
    run;"
)

olesas.Submit ("data;
    %let folder=" & folder & ";
    %inc '&folder\print.sas';
    run;"
)

'Set up the SAS© session running visible during run time
olesas.Visible = True

'The following two lines of code is for ending the SAS© session
olesas.Quit
Set olesas = Nothing

Exit_Run_sas_Click:
Exit Sub

'The following three lines of code provides the error message
'if there is any errors

Err_Run_sas_Click:
    MsgBox ("Please check folder, autoexec.sas, and print.sas")
    Resume Exit_Run_sas_Click
End Sub

Following is the attached SAS© program, print.sas:

    %inc '&folder\autoexec.sas';

    Proc Contents data=&lib..&dsn (keep=memname name type label format) noprint;
    run;

    Proc Export data= &dsn outfile = "&folder\dsn.xls" dbms=excel2000 replace;
    Run;

CONCLUSION
Automation with a GUI can be implemented using MS-Office embedded Visual Basic tools or the Visual Basic software.

By using automation with a GUI, the SAS session and its run-time activities can be set up to take place behind the scenes. Users do not necessarily need to be knowledgeable of SAS to do data manipulation, statistical analysis, database maintenance, figure or table report generation, documentation, or delivery of clients' reports through an Intranet or the Internet by integrating SAS and MS-FrontPage.

This can improve efficiency, reducing the skill requirements to do such tasks and increasing the utilization of SAS programmers.

REFERENCE

ACKNOWLEDGMENTS
The author and co-author of this paper sincerely thank Doug Lassman for his taking precious time to review the draft.
CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the author at:
Rubin Nan
PRA International, Inc.
16400 College Boulevard
Lenexa, Kansas 66219
Work phone: (913)-577-2882
Fax: (913) 599-0344
Email: NanRubin@praintl.com

David Mullins
PRA International, Inc.
16400 College Boulevard
Lenexa, Kansas 66219
Work phone: (913)-577-2946
Fax: (913) 599-0344
Email: MullinsDavid@praintl.com

SAS and all other SAS Institute Inc. product or service names are registered trademark or trademarks of SAS Institute Inc. in
the USA and other countries. ® indicates USA registration.

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