Paper 32-28

Using IOM and Visual Basic in SAS® Program Development

Greg Silva, Biogen, Inc., Cambridge, MA

ABSTRACT

When SAS Institute added the Integrated Object Model (IOM) to the SAS System's base product in version 8, they opened the door to SAS applications development using Visual Basic (VB). Data set manipulation, SAS code execution, and data binding are now part of the tool set available to the SAS/VB programmer. This paper will describe some of the features of SAS that are available through the use of IOM, and show some applications that use IOM with VB.

INTRODUCTION

Initial desktop development of applications outside of the SAS System relied on PC batch files. These were usually simple "if-then-else run a SAS program" scripts that simplified the parameter selection of SAS macros.

With the advent of the integrated technologies in SAS, developers can greatly enhance the user input into SAS, or even have SAS running within applications. This paper will show some examples of VB programs that can be used with IOM to run SAS sessions.

The intended audience for this paper is any developer interested in building front-ends for SAS processing, or those people familiar with VB who want extend the use of VB into SAS data manipulation. The information presented in this paper is intended for programmers who may have heard of IOM, but are not that familiar with it.

I would like to add a caveat about the code examples used in this paper. All of the examples were derived from the SAS Institute's online IOM documentation, or by playing around with the objects and their methods. The code works, but is not necessarily the most efficient.

WHAT IS THIS THING CALLED IOM?

The IOM component of the SAS integrated Technologies product transforms the SAS system into a server for processing data from client locations. By setting up references to the SAS IOM components in your VB program, you can do most of things that you do with SAS in batch.

Local user access to IOM is as simple as not specifying a server when the Workspace is opened. For some programs, this seems to help with the execution speed. Using this method assumes that you have a copy of the SAS System on your local machine.

Server access requires more hardware to run. The documentation provided by SAS Institute goes into detail about running IOM bridges and using LDAP for authenticating user IDs and password. For the purpose of this paper, I will only talk about local access.

HOW DO YOU SET IT UP?

There are two steps you have to take before using IOM in a VB program. The first is to get the components from SAS Institute. The second is to add the references for IOM to your VB application.

INSTALLING THE COMPONENTS

SAS Institute's Integrated Technologies is available for PCs, and can be downloaded from SAS Institute's Website. The documentation states that it needs version 8.2 to run, however all of the examples in this paper were run using version 8.1 of the SAS System and the downloaded Integrated Technologies package.

The downloaded package is a self extracting exe file that will copy the needed components to specific folders on your PC. Be aware that you may have to reboot your machine if some of the Microsoft components on your machine are not up to date.

GLOBAL REQUIREMENTS

When you use IOM, you have a number of options for running SAS. Simple includes, stored processes, or even dynamically running programs can be accomplished with a Workspace defined in IOM.

Each one of these methods for running SAS will be explained in this paper, but before you start running SAS, you have to tell VB what SAS is. This is done by referencing components of the SAS system when a program is written.

In order to get started using SAS Institute's Integrated Technologies, the following references have to be added to your VB project:

1. SAS: Integrated Object Model (IOM) (SAS System 8.2) Type Library

2. SASWorkspaceManager 1.1 Type Library.

These are both used to allow the communications between VB and the SAS System. If you do not see these as options in your reference list, then you have not installed the integrated technologies package correctly.

In the examples that follow, I have used global references to objects. Although this is not considered a best programming practice, I have done it to avoid having to set or pass objects for each function or subroutine.

To make variables or objects global, you have to reference them outside of a subroutine or function. To use the examples in this paper, the following lines should be at the start of your program.

```
Dim obSAS    As SAS.Workspace
Dim obWM     As New _
          SASWorkspaceManager.WorkspaceManager
```

Please note that I have attempted to add code that can be run without changes. In order to fit the VB code in the format of this paper, I have had to split some of the text. In the example above, the "_" is used in VB to show a continuation line. This will be used as needed in the remaining examples.

HOW DO YOU USE IT?

Now that you have everything set up on your system, and your VB application knows about IOM, you are ready to start coding. All of the features of IOM can be used on a single subroutine in VB, but for reuse and readability, you should consider writing separate subroutines or functions for each of the IOM tasks you expect to use.

The following sections will cover some useful parts of IOM, and present them as re-useable routine (either functions or subroutines). As you develop more VB applications that use IOM, these routines can be carried over to save time.

STARTING SAS

To use the Workspace provided in IOM, you have to create a SAS object. This can be done anywhere in the code, but is best done as a stand alone subroutine or function.

The following function creates a SAS Workspace. For this example, the obSAS object that is created is referenced in the global Dim area. It could have been returned from the function if there was not a global reference.

```
Public Function Start_SASjob()
   Dim xmlInfo As String

   ' Create a local SAS workspace.
   Set obSAS =

   obWM.Workspaces.CreateWorkspaceByServer _
                  ("", VisibilityProcess,_
                   Nothing, "", "", xmlInfo)
End Function
```

After calling this function, a SAS Workspace object called obSAS will be ready for use. Since it was previously declared as global, it is available outside of the Start_SASjob function.

In the code above, the first parameter passed to the method CreateWorkspaceByServer is blank. This indicates that a local instance of SAS exists that will act as the IOM server. If you did not have the SAS System installed on your machine, this code would not work. To use IOM in non-local mode, you need to specify a server.

STOPPING SAS

Since SAS will continue running a Workspace until you stop the process (remove the object), you need to have a routine to end SAS. The following function will stop a SAS IOM session.

```
Public Function End_SASJob()
   obWM.Workspaces.RemoveWorkspaceByUUID _
obSAS.UniqueIdentifier
   obSAS.Close
End Function
```

Note that the SAS Workspace object (obSAS) has a unique identifier. This allows you to have multiple SAS sessions running within your application. You have to keep track of each of their unique Ids.

In the world of Visual Basic (and Java and C++) you need to be aware of the persistence of objects. In the sample code above, the first line removed the workspace from use. The second line ("Invoke the close method on the Workspace object called obSAS") is what actually ends the SAS session.

USING AN INCLUDE FILE

One of the simplest ways to run a SAS program using IOM is to include the SAS source, and send it to be run. In the following example, a string containing valid SAS source is stored in an array. The string consists of a %include and the full file name of the SAS program to be run.

The name of the program is passed to the function, the object for the LanguageService is set based on the Workspace object (obSAS) that was created previously. The string is stored in an array to allow the SubmitLines method to be invoked.

```
Public Function Submit_SASInclude (strCmd As_
                                    String)

   Dim obLS        As SAS.LanguageService
   Dim arSource(1) As String

   Set obLS = obSAS.LanguageService

   arSource(0) = "%include '" & strCmd & "';"
   obLS.SubmitLines arSource
End Function
```

An alternative method would be to read all of the SAS program into an array and pass the array to the method. The advantage to including a file is that the SAS source code can be modified without recompiling the Visual Basic code.

USING A STORED PROCESS

Stored processes are like macros without the parameters. The SAS source would be a macro call followed by an invocation of the macro. Parameters are passed to the macro through %let statements before the macro source.

With store processes you have to give SAS three pieces of information: The repository (directory name), source (program name), and command (macro values). In the case of the stored process, the macro variables are defined as:

Variable = Value

No %let or semicolon is needed to set the values.

In the following example, a function passes the three values needed to invoke a stored process, checking for a null string in the command. If the command is null, a dummy variable is generated to allow the execute method to be invoked without errors.

```
Public Function Submit_SASProcess( _
                resp_folder As String,_
                source_file As String,_
       Optional command_text As String)

   Dim obStoredProcessService As _
       SAS.StoredProcessService

   Set obStoredProcessService = _
obSAS.LanguageService.StoredProcessService

   obStoredProcessService.Repository = _
      "file:" & resp_folder

   If (command_text = "") Then
      command_text = "__dummy_ = NONE"
   End if

   obStoredProcessService.Execute _
         source_file, command_text
   End If
```

```
End Function
```

DYNAMIC CODING

Using the Start_SASJob function, and passing SAS source code directly to the Workspace through the SubmitLines method allows you to dynamically run SAS programs.

Through a user interface in VB, SAS source code could be "built" to be submitted through IOM. Simple processes such as building a parameter list and calling a macro could easily be incorporated into an application.

The following code is a simple function to submit a line of SAS source code to the IOM server. Notice that this is simply a variation on the include file code, but it allows the developer to pass multiple lines to the IOM server.

```
Public Function Submit_SASCode (strSASCode
                                As_ String)

   Dim obLS        As SAS.LanguageService
   Dim arSource(1) As String

   Set obLS = obSAS.LanguageService

   arSource(0) = strSASCode
   obLS.SubmitLines arSource
End Function
```

Some of the work we are currently doing involves querying user's for macro parameters, and then adding extra features to the resulting code. This includes ODS destinations, and post processing output by invoking Microsoft Word and polishing the output format through VBA.

We have found this last method very useful. By removing macros from the Microsoft Word templates, we can control when and where they are used. An application we are developing allows users to select multiple SAS source files, run them, check the logs for errors, and then convert the SAS output (.LST files) to Word documents with an optional watermark.

CURRENT EVENTS

To add event driven capabilities to your IOM components, use **WithEvents** with the LanguageServices object.

```
Dim WithEvents obLS As SAS.LanguageService
```

Once the events are monitored, you can set up routines that will be invoked when the events occur. For instance, in IOM you can monitor the beginning and end of data steps and procedures.

The following example is a subroutine that is invoked automatically when a data step has completed. In this case, when a data step is completed the log file is sent to a window in the system, and a message is flashed on the screen.

```
Private Sub obLS_DatastepComplete()
   Call DumpLogToWindow
   Msgbox "DATA step has completed"
End Sub
```

The next subroutine is invoked automatically when a procedure completes. In this case, when a procedure is completed, the log file is sent to a window in the system, and a message is flashed on the screen. If the procedure is a PROC PRINT, then the output is sent to a window.

```
Private Sub obLS_ProcComplete(ByVal Procname_
            As String)
      Call DumpLogToWindow
      Msgbox "PROCEDURE has completed"

      If Procname = "PRINT" then
         Call DumpListToWindow
      End if
 End Sub
```

PUTING IT ALL TOGETHER

Using all of the sample functions and routines together, a simple VB program can be produced that will run SAS source entered by the user. In the code segment below, SAS is started, code is submitted, the user is given feedback, and the SAS System is terminated.

```
   ' Global refs to Workspace manager stuff.
   ' Assume IOM and Workspace references in VB.
   Dim obSAS   As SAS.Workspace
   Dim obWM    As New _
            SASWorkspaceManager.WorkspaceManager

   ' Tell VB that events should trigger Subs.
   Dim WithEvents obLS As SAS.LanguageService

Sub Main()
   ' Load Workspace for processing.
   Start_SASjob()

   ' Process something.
   Submit_SASCode("data a; message = 'Hello,_
                  World!'; run; proc print;_
                  run;")

   ' NOTE: At end of DATA step the event will
   '       be triggered to display
   '            "DATA step completed"
   '       followed by the event trigging the
   '             "PROCEDURE has completed".
   '       message. This would be followed
   '       a messagebox with the message:
   '             "Hello, World!"
   '       (Assuming your DumpListToWindow
   '       routine did that.)

   ' Close Workspace.
   End_SASjob()
End Sub
```

At this point the Workspace would no longer exist, and you could end your VB program. You could also add any other code before you end the IOM session, and all temporary data sets and macro variables would be available for use.

The example above references one of the other features of IOM we use: the ability to redirect SAS logs or SAS output to an array of strings in VB. By using the FlushLogLines or FlushListLines method of the SAS.LanguageService object, you can store or display log and list information from your IOM session.

BUT WAIT… THERE'S MORE!

This paper is an attempt to show some of the basic methods that IOM and SAS Institute's Integrated Technologies can be used to encapsulate most of the SAS System's functionality into VB. There are many more components that have not been addressed here. The following are some of the other ways that IOM can be used:

1. Data Visualization. Using the Active Data Objects in VB, you can build “data aware” components to easily display data set information on the screen.

2. Multitasking applications. Multiple SAS Workspaces can be run at the same time. This would allow for multitasking applications, but you would have to keep track of when each process finished.

3. Direct data set manipulation. By using other methods in IOM, you can add, delete and modify records in data sets without using any SAS code.

4. True Client/Server applications. An IOM server can be set up with any number of clients. The server would “serve up” SAS Workspaces, while the clients (PCs without SAS installed) would use these Workspaces to process data.

CONCLUSION

SAS has enhanced its application toolbox by allowing developers to work in compiled languages, but still have access to the core language, as well as interfaces to the data set. By using Visual Basic on a PC, or Java for cross-platform applications, the developer can make use of standard GUI components for the front-end, while relying on SAS for data manipulations on the back-end.

REFERENCES

I found the following are three sources very helpful in getting up to speed in VB and IOM.

1. “Developing Windows Clients” from the Distributed Objects section of the SAS Website:

http://www.sas.com/rnd/itech/doc/dist-obj/winclnt/index.html

2. If you want a great book for learning Visual Basic, as well as an outstanding source of Visual basic code try:

   Programming Microsoft Visual Basic 6.0
   by Francesco Balena
   Microsoft Press ISBN 0-7356-0558-0

   This book contains a huge amount of VB information. It has a great section on ADO (Active Data Objects) that helps set up your VB program to be “data aware”. A companion CD contains all the source code from the book.

3. SAS-L. Although I did not ask for information on IOM, I searched the archives for samples and pointers on using VB with IOM.

CONTACT INFORMATION

For any comments or questions, feel free to contact the author.

Greg Silva
Biogen, Inc.
14 Cambridge Center
Cambridge, MA 02142

Work Phone: (617) 679-2560
Email: Greg_Silva@biogen.com