ABSTRACT
Base SAS® Integration Technologies opens the door for applications written in a variety of languages to access the power of SAS. Integration Technologies allows Visual Basic programmers to easily combine industry standard tools along with SAS to quickly create powerful applications. This paper presents an introduction to the tools, techniques, and strategies that VB programmers can employ in their own applications.

INTRODUCTION
One question frequently asked is “Which programming language should I use to access SAS with?” The answer is to use whichever one you want. Thanks to support of both COM and Corba objects from SAS, a wide array of languages can access SAS. This paper focuses on using the SAS COM™ objects from Visual Basic. Most of the discussion here also applies to Visual C++ programs too. Visual C++ programs can access COM interfaces in almost the same way Visual Basic programs do; the main difference is the language syntax.

The SAS System for Windows has had an OLE automation interface available since version 6.12. That interface is different from the Integration Technologies interfaces in several ways:
- The Integration Technologies Object Hierarchy provides many more features and functions. The old OLE automation interface is a single interface; Integration Technologies provides a hierarchy of interfaces.
- A VB application can remotely use Integration Technologies Objects running on most hardware platforms that SAS supports, including Windows, OS/390, and Unix. The OLE automation interface is only available on Windows.
- The Integration Technologies Objects allows SAS programs to run asynchronously. This allows your VB application to continue to run while SAS is performing analysis.
- The root object for the Integration Technologies objects has a progid of “SAS.Workspace”; the progid for the OLE automation object is “SAS.Application”.

Visual Basic itself actually has three different variations: Visual Basic (used in Visual Studio), Visual Basic for Applications (used as the macro language for Word, Excel, PowerPoint, Visio, Access, Outlook 2000), and Visual Basic Scripting (used in Outlook 98, Windows Scripting host, and Active Server Pages). The support that SAS provides for COM objects is the mechanism by which all of these variations access SAS. Each of these variations has slight differences which impact how SAS (or, any COM object) is used. It is possible to use all the SAS COM objects from each of these variations.

Integration Technologies consists of several pieces, all of which can be useful to VB applications:
- IOM is the nickname for the set of COM objects that SAS makes available. For VB applications, this is the most commonly used tool. This and the tools SAS developed to work with IOM is the focus of this paper.
- LDAP Directory Integration. LDAP is an industry standard for a distributed store of information. VB programs can access an LDAP directory using Microsoft’s ADSI. You can think of LDAP as a database available through TCP/IP. It is generally assumed that an organization will have a single LDAP server that all machines in that organization share.
- Publication/Subscribe describes a set of functionality that allows you to make SAS output available (publish) to those who may be interested in it (subscribers.) This consists of IOM interfaces, SAS Language call statements, and LDAP objects that work together. To understand the underlying technology used for Publication/Subscribe, you will need to understand Result Packages. Result Packages are discussed below.
- Message Queuing allows SAS output to be written to a message queue, so the client does not have to stay around while the server analyzes the output. Integration Technologies delivers support for Message Queuing in the form of SAS Language call statements. Use of Message Queuing is an advanced topic and will not be covered in this paper.

IOM
IOM stands for Integrated Object Model. It is an object model because it exposes a set of objects that can be programmed to. It is integrated in the following ways: first, the objects are a part of the SAS system; and second, it is easy to make these objects a part of an application.

Currently available objects are centered on two goals: submitting SAS Language programs to SAS and getting output back from SAS.

All IOM objects are designed for distribution. There is extensive use of arrays, so all the output that may be needed is returned all at once instead of across multiple calls. Integration Technologies still depends on SAS Language to perform Data analysis. This use of SAS Language to submit SAS jobs is a very good thing for distribution; it allows custom complex tasks to be performed in a single network round-trip.

The IOM interfaces are available with Base SAS on Windows; but use of this is restricted to local COM unless you have licensed Integration Technologies. The Integration Technologies license grants you the right to receive IOM calls through DCOM and TCP/IP (with the IOM Bridge for COM.)

The root object for the IOM hierarchy is the Workspace. Those familiar with the Microsoft Word, Excel, Visio, or PowerPoint object models should be familiar with the Application object. A SAS Workspace is very similar to the Application object; but one big difference is that we can have multiple Workspace objects in a single operating system process. You will typically only use one Workspace at a time from any given SAS process. Each Workspace has its own WORK library, and it is not possible to share the WORK library between Workspaces.

Some of the methods available on the IWorkspace interface are:
- DataService – returns an interface used to manipulate SAS librefs. The DataService also contains interfaces for reading and writing SAS datasets; but VB programs should not use these data access interface directly. VB applications should use the IOM Provider, which actually uses this DataService interface to read and write the data.
- FileService – returns an interface used to read, write, and enumerate files and filerefs running on the SAS Server.
- GetApplication – returns a custom interface created in SAS/AF®.
LanguageService – returns an interface used to submit SAS code to SAS, and get the output back.

The Workspace Manager should be used to get a Workspace. More details about the Workspace Manager are discussed in the “Workspace Manager” section of this paper. For the examples in this section, assume the following code has already executed (note the use of the underscore character throughout the VB code in this paper; this means the line continues):

```vba
Dim obWsMgr As New _
    SAS.WorkspaceManager.
Dim obSAS As SAS.Workspace
Dim xmlInfo As String
' This creates a SAS Server running on the local machine
Set obSAS = _
    obWsMgr.Workspaces.CreateWorkspaceByServer _
    ("", VisibilityNone, nothing, "", ", _
    xmlInfo)
```

There are two ways to submit SAS code to the SAS server. One way to submit SAS code is to use the LanguageService.Submit method. This requires all the SAS code to exist in the VB code (unless the VB code reads it out of a file and then passes it to SAS.)

```vba
obSAS.LanguageService.Submit _
    "data a; do x= 1 to 10; y=x*x*x; " & _
    "output;end;run;"
MsgBox obSAS.LanguageService.FlushLog(100000)
```

The other way is to use the StoredProcessService. This requires the SAS code to exist in a known directory on the machine where the SAS Server is running. There is no limit to the number of parameters that can be passed from the VB program to the SAS Stored Process. Separate multiple parameters with a space. In the example below, the loopTimes macro is entered as a parameter from the VB program

```vba
' Run the SAS program at c:\temp\looper.sas
Dim obStoredProcedure As _
    SAS.StoredProcessService
Set obStoredProcedure = _
    obSAS.LanguageService.StoredProcessService
obStoredProcedure.Repository = _
    "file:c:\temp"
MsgBox obStoredProcedure.Execute "looper", _
    "loopTimes=10"
MsgBox obSAS.LanguageService.FlushLog(100000)
```

And here is the stored process in c:\temp\looper.sas:

```sas
%let loopTimes=3;
*ProcessBody;
    data a;
    do x= 1 to &loopTimes;
        y=x*x*x;
        output;
    end;
run;
```

Note the use of the %ProcessBody; statement. This instructs the StoredProcessService to replace that line with macro definitions it receives on the call to obStoredProcedure.Execute. In this case, the macro inserted is %LET LOOPTIMES=10; The macro definitions that occur before the ProcessBody statement are default values; if the VB program does not specify macro values, then the values specified before the ProcessBody statement will be used.

Use of the StoredProcessService encourages good programming practices by helping to separate the SAS code from the VB code. This separation makes maintenance of both the SAS code and the VB code easier.

Calls to the Execute and ExecuteWithResults methods as well as the LanguageService.Submit method are influenced by the LanguageService.Async property. If this property is false, calls to Execute, ExecuteWithResults, and Submit will not return until SAS has finished running all the code submitted. If the Async property is true, calls to these three methods will return immediately.

Setting Async to true is a great way to submit multiple jobs to multiple SAS servers at the same time. The events available on the LanguageService can be used to detect when processing has completed, to get a hint of how much progress has occurred, or to get notification of an error in the submitted code.

The easiest way to determine if an error has occurred in the submitted SAS language is to listen for the StepError event:

```vba
Public WithEvents obSASLanguage As _
    SAS.LanguageService
Private Sub obSASLanguage_StepError()
    ' An error has occurred. Dump the log
    Debug.Print obSASLanguage.FlushLog(100000)
End Sub
```

After the SAS language has completed processing, most applications will want to get the output. There are many types of output available. The IOM Data Provider can be used to get raw data. The FlushList or FlushListLines methods of the LanguageService can be used to get the contents of the Output window (note that the Output window is not visible when running SAS through IOM.) To get ODS output, have SAS generate the output to a fileref, then use the FileService to retrieve it. Here’s an example of using the FileService. The SAS code can be submitted through the LanguageService.Submit or StoredProcessService.Execute methods:

```vba
Dim obFileref As SAS.Fileref
Dim obTextStream As SAS.TextStream
Dim obFileSystem As New _
    Scripting.FileSystemObject
Dim obFile As Scripting.TextStream
Dim obTextStream As SAS.TextStream
Set obFile = obFileSystem.CreateTextFile _
    ("c:\temp\sasoutput.htm", True)
```

The VB code reads the generated ODS output from the same temporary fileref, fref:

```vba
' The point of this code is to generate some ODS output on the SAS server, and transfer it to a file on the client.
' You can just have SAS write the output directly to a file and not use the SAS FileService
Dim obFileref As SAS.Fileref
Dim obTextStream As SAS.TextStream
Dim obFileSystem As New _
    Scripting.FileSystemObject
Dim obFile As Scripting.TextStream
Set obFile = obFileSystem.CreateTextFile _
    ("c:\temp\sasoutput.htm", True)
```
"filename fref TEMP;" & _
"ods html body=fref;" & _
"proc corr data=sashelp.class;" & _
"run;" & _
"ods html close;"

' This fileref, fref, was created in the SAS
' Language we submitted.
Set obFileref = _
obSAS.FileService.UseFileref("fref")
Set obTextStream = obFileref.OpenTextStream(StreamOpenModeForReading, 10000)
sOdsOutput = obTextStream.Read(100000)
While (Len(sOdsOutput) > 0)
' Do something with the read text here
obFile.Write sOdsOutput
sOdsOutput = obTextStream.Read(100000)
Wend
WebBrowser1.Navigate "c:\temp\sasOutput.htm"

Another alternative for getting output from SAS is to use the
ResultPackageService. The ResultPackageService interfaces are
available from Workspace.Utilities.ResultPackageService. There
are two ways to create a ResultPackage: using the
ResultPackageService interfaces, or, more typically, use SAS
Language CALL statements to put together a ResultPackage. A
ResultPackage can consist of any type of SAS output, including
files, filerefs, datasets, ODS output, images, and more. Once a
ResultPackage is created, it can be written (published) out to a
variety of destinations, including an archive, an email recipient, a
webDAV directory, and more. An Archive is really just a
compressed (ZIP) file. It provides a way to easily store SAS
output. The IOM ResultPackageService can be used to read a
ResultPackage (whether it is in an archive or not.) Here's some
SAS code that creates an archive:

%macro CheckRC(rc);
  if rc ne 0 then do;
    msg = sysmsg();
    put msg;
    ABORT;
  end;
%mend;

data _null_;
call PACKAGE_BEGIN(pid, desc, nameval, rc);
%CheckRC(rc);

call INSERT_FILE(pid, 'FILEREF:fref',
  "TEXT", "text/html", "Some ODS Output",
  ", ', rc);
%CheckRC(rc);

/* Nothing in the package actually gets
* written out until we call publish.
* So, if you modify any filerefs after
* calling insert but before calling
* this, then you will get the
* modified filerefs.*/
call PACKAGE_PUBLISH(pid, "TO_ARCHIVE", rc,
  "archive_path, archive_name", "c:\temp",
  "archive*");
%CheckRC(rc);

/* You could call PACKAGE_PUBLISH as many
* times as you want for any given package,
* as long as you

Sample VB code that reads the ResultPackage.
Dim props() As String
Dim obResultPackage As SAS.ResultPackage
Dim obFileEntry As SAS.ResultPackageFileEntry
Dim obRPS as SAS.ResultPackageService
Set obRPS = _
obSAS.Utilities.ResultPackageService
Set obResultPackage = _
obRPS.BrowseResultPackage( _
  "ARCHIVE", "c:\temp\archive.spk", props)
Set obFileEntry = obResultPackage.GetEntry(0)
Set obTextStream = _
obFileEntry.Open(100000)
sOdsOutput = obTextStream.Read(100000)
 While (Len(sOdsOutput) > 0)
' Do something with the read text here
obFile.Write sOdsOutput
sOdsOutput = obTextStream.Read(100000)
Wend
WebBrowser1.Navigate "c:\temp\sasOutput.htm"

The above example knows what is written to the SAS package, so
it uses an absolute index into the package to read the HTML. The
ResultPackageService interfaces have enough functionality to write
a more generic application that can discover the information in a
ResultPackage, and present it in a reasonable way. This is exactly
what the SAS SPK Reader application does. The SPK Reader is
included with SAS/Integration Technologies, and can be freely
redistributed.

If the output you're interested in is in a SAS Dataset, you will want
to use the SAS DataService IOM interfaces. You generally would
not use these interfaces directly; instead, you should read and
write SAS Datasets using the IOM Data Provider.

In order to initially get a SAS Workspace, use the SAS Workspace
Manager.

THE SAS WORKSPACE MANAGER
The SAS Workspace Manager is an ActiveX control that performs
three functions:
1. Establishes a connection with SAS, returning a SAS
   Workspace.
2. Provides pooling of SAS Workspaces, useful in a web
   environment where a SAS Workspace is used briefly and
   then discarded.
3. Manages connections with SAS, allowing you to obtain
   previously created SAS Workspaces.

There are two different techniques to use when establishing a
connection with SAS: unadministered (where all the connection
information is provided in the VB code) and administered (where all
the connection information is provided in either a file or LDAP.) The
connection information includes what machine to connect to, the
protocol (bridge or COM), the port or service name (for bridge
connections), and the userid and password (for bridge
connections). There are also parameters that control the security
of the connection. Both DCOM and the IOM Bridge for COM can
support encrypted sessions.
To establish a connection to a remote SAS machine in an unadministered environment:

```vba
Dim obWSMgr As New _
SASWorkspaceManager.WorkspaceManager
Dim obSAS As SAS.Workspace
Dim xmlInfo As String
Dim obServer As New _
SASWorkspaceManager.ServerDef

Set obServer = _
obWSMgr.Workspaces.CreateWorkspaceByServer _
("", VisibilityNone, obServer, ",", ",", _
xmlInfo)
```

The xmlInfo parameter is a returned string that contains details about which server definition was actually used. It is possible to administer multiple machines that could be connected to. This is used for failover; not load balancing. If one of those connections fail, another is used until either all connections fail or one works. Delivering this information in XML allows it to be easily parsed, thereby providing many alternatives for presenting the error to the user. The most common reason for a failed connection is an incorrect username/password. If all connection attempts fail, the CreateWorkspaceByServer or CreateWorkspaceByLogicalName method fail, and the xmlInfo is delivered in err.description instead of xmlInfo.

The connection information necessary to obtain a SAS Workspace can be administered into either an LDAP server or a text file. SAS/Integration Technologies ships with the Integration Technologies Administrator, which can create the appropriate LDAP entries for you. If you choose a text file, administration will be left up to you and your favorite text editor. To establish a connection to a remote SAS machine where the connection information has been administered into LDAP, declare the same variables, but instead of calling CreateWorkspaceByServer, call

```vba
With obWSMgr.Workspaces
Set obSAS =
.CreateWorkspaceByLogicalName("", _
VisibilityNone, "LogicalName", _
"cn=Dan,dc=sas,dc=com", xmlInfo)
```

It is possible to create a SAS Workspace using the standard VB code:

```vba
Dim obSAS as new SAS.Workspace
```

However, by using the workspace manager to create connections with administered connection information, you can more easily change the machine where SAS is running. This may be necessary as an application grows in popularity, and gets more users.

The pooling capabilities of the Workspace Manager are new in Version 8.2 of SAS. As of this writing, 8.2 has not shipped yet, so the pooling mechanisms may change. The Workspace Manager will allow you to use either COM+ or the Integration Technologies Administrator to administer the pool configurations. In both cases, you get a PooledWorkspace object. This object supports a single property, Workspace, which is the SAS Workspace that has been pooled. You must maintain a reference on the PooledWorkspace object for as long as you want to use it. When you release the PooledWorkspace object, you can no longer use the workspace.

The third reason to use the Workspace Manager is if you want to pass the SAS Workspace interface to another object, or to yourself at a later time. The IOM Data Provider uses this mechanism to make a connection to SAS.

**THE IOM DATA PROVIDER**

The IOM Data Provider is an OLE/DB data provider that uses the IOM DataService interfaces to read and write data. You will use Microsoft ActiveX Data Objects (ADO) to read and write data from a VB application. ADO is really a layer on top of OLE/DB that enables applications to use OLE/DB providers.

ADO and SAS use a few terms for the same things differently:

<table>
<thead>
<tr>
<th>SAS Term</th>
<th>ADO Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset</td>
<td>Recordset</td>
</tr>
<tr>
<td>Observation</td>
<td>Record</td>
</tr>
<tr>
<td>Variable</td>
<td>Field</td>
</tr>
</tbody>
</table>

To use the IOM Data Provider, you first create a connection with SAS. You can do this using either the unadministered or administered techniques outlined in the Workspace Manager section above. However, one important difference is that the Visibility parameter of either CreateWorkspaceByServer or CreateWorkspaceByLogicalName must be set to VisibilityProcess. This tells the WorkspaceManager to keep track of the created workspace, and to allow any request that comes from the same process (even if in a different DLL or thread) to use that workspace you created. This also means that when you are done using the workspace that you should notify the WorkspaceManager that it can release the Workspace by calling WorkspaceManager.Workspaces.RemoveWorkspace. Here’s some sample code that creates a workspace, and opens the ADO connection:

```vba
Set obSAS = _
obWSMgr.Workspaces.CreateWorkspaceByServer _
("",VisibilityProcess,obServer,"","", _
xmlInfo)
```

Now that we have a connection, we can open one or more Recordsets. To open a recordset, you can use an SQL string to run a query, or you can use a dataset name. Then you can do all sorts of things with the recordset; in this sample we’ll bind the recordset to a Microsoft Chart Control (which we created a reference to and an instance of in the VB development environment):

```vba
Dim obRS As New ADODB.Recordset
obRS.Open "sashelp.class", obConnection, _
adOpenDynamic, adLockReadOnly, _
adCmdTableDirect
Set MSChart1.DataSource = obRS
```

**THE INTEGRATION TECHNOLOGIES CLIENT**

There are files that need to be installed and registered on the client machine in order to run against a SAS Server. The Integration Technologies client is a self-installing executable that installs and registers the files necessary to run SAS across the network. This includes the type libraries (which COM uses for marshalling, and VB uses to resolve constants and for intellisense), the IOM Bridge for COM, the SAS Workspace Manager, Scripto, and the IOM Data Provider.

The Integration Technologies client self-installing executable is called inttech.exe. You can include this as part of your install program when you create a VB application. Using this install instead of writing your own ensures that other applications that rely...
on this same functionality can share the installed pieces. Also note, as with all ActiveX controls, that only one control of any given type can be registered at any time. So, if you install these shared controls in your own program directory, all other applications that use the control will start using your control. And when you uninstall your application, the registry entries would be removed, and the other applications that depend on these controls would fail.

When you install Base SAS running on a Windows operating system, the Integration Technologies Client is installed along with SAS. If you write a VB client that accesses SAS across the network, and SAS is not installed on the client machine, you will need to install the Integration Technologies Client. You can install the Integration Technologies client on as many machines as you want.

**THE IOM BRIDGE FOR COM**

Integration Technologies allows you to transparently make COM calls to a SAS server running locally, on another Windows machine, or on a Unix or OS/390 machine. When connecting your client to SAS running on Windows, you can use COM or DCOM without involving the IOM Bridge for COM. The key to getting COM interfaces to SAS running to either Unix or OS/390 is The IOM Bridge for COM. It is possible to use the IOM Bridge for COM to connect to a Windows machine also.

### Server Process  
### Client Process

The IOM Bridge for COM is a middleman, where on one side it presents COM interfaces for communication with your VB application. On the other side it presents a TCP/IP connection for communication with SAS. Then in SAS there is another middleman that communicates with TCP/IP and converts that into internal SAS calls. Once the call is within SAS, SAS doesn’t know (and doesn’t care) if the call came from a local COM connection, a DCOM connection, or a Bridge connection.

This whole process is remarkably fast. This mechanism is capable of making on the order of 1000 calls per second (where no processing takes place in SAS).

The IOM Bridge for COM is implemented as an Active X Control, in SASComb.dll. However, you should never make any calls directly on the SASComb interface, and there is no documentation on how to make direct calls to SASComb.dll. Instead, you should use the SAS Workspace Manager to obtain interfaces that connect to SAS through the IOM Bridge for COM.

**SCRIPTO**

There are several limitations to VBScript which Scripto can help you work around. Scripto is an Active X control written by SAS for use in VBScript applications. Use of Scripto is not necessary in all VBScript applications. The main reasons you may need to use it are:

1. You want to call a method that returns more than one value.
2. You want to call a method that uses arrays.

Here’s a sample call to SAS that uses Scripto:

```vb
' Scripto will do 2 things for us here:  
' 1) It will convert the 3 returned arrays to  
' arrays of variants instead of arrays of  
' longs so vbscript can handle them

set obSAS = CreateObject("SAS.Workspace")
set obScripto = _
    CreateObject("SASScripto.Scripto")
obSAS.LanguageService.Submit _
    "proc options;run;"
obScripto.SetInterface obSAS.LanguageService

' obSAS.LanguageService.FlushLogLines 1000, _
' carriageControls, linetypes, logLines
Dim flushLogLinesParams(3)
flushLogLinesParams(3) = 1000
obScripto.InvokeMethod "FlushLogLines", _
    flushLogLinesParams

' flushLogLinesParams(0) now has logLines
' flushLogLinesParams(1) has lineTypes
' flushLogLinesParams(2) has carriageCtrls

' This prints the first line
wscript.echo flushLogLinesParams(0)(0)
```

Note that use of Scripto is only necessary when you are using VBScript. It is not necessary in VB or VBA. You should avoid using Scripto unless you must; it introduces additional overhead to each method call that uses it. It is possible to mix calls on the same interface between making them through Scripto and making them as you normally do. So, it is easy to only use Scripto on those calls that require it.

**CONCLUSION**

By combining Visual Basic and SAS, it is easy to create powerful applications to solve a variety of problems. All of the tools introduced here have much more functionality than was presented here. Be sure to check out the Integration Technologies web site for more details and new developments.

**REFERENCES**

From this URL, [http://www.sas.com/rnd/itech/intro.html](http://www.sas.com/rnd/itech/intro.html)  
Click on “Library”, then “Windows Client Development”. The following documents are available:

- “Programming with Visual Basic ”
- “Using the Workspace Manager”
- “Using the IOM Data Provider”
- “Using VBScript”

Source code presented in this paper is available on the SAS web site at: [http://www.sas.com/rnd/itech/](http://www.sas.com/rnd/itech/)

Base SAS is a registered trademark of SAS in the USA and in other countries ® indicates USA registration.

**CONTACT INFORMATION**

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

- Daniel Jahn  
  SAS Institute  
  1 SAS Campus Drive  
  Cary NC 27513  
  Fax: 919 677-4444  
  Email: Dan.Jahn@sas.com  