

Communicating with SAS using SAS Integration Technologies

Lynn Pope, Amadeus Software Limited



ABSTRACT

SAS Integration Technologies enables programmers to develop component-based applications that integrate SAS features into your application, using any programming language that supports distributed object programming.

An essential aspect of Integration Technologies is its use of the Integrated Object Model (IOM) for client/server communication. IOM is a set of Component Object Model (COM) objects that SAS makes available, and is the most commonly used tool for VB applications.

This paper will focus on how applications can take advantage of SAS Software's rich set of features and ability to access a wide variety of data via SAS Integration Technologies. This will be demonstrated by developing an open client application using the Visual Basic programming language and IOM communication.

This paper assumes a general knowledge of Visual Basic, COM and SAS, but not an in-depth knowledge in any of these areas.

INTRODUCTION

SAS Integration Technologies (IT) enables open client access to SAS software, allowing developers to integrate SAS with industry standard tools to create applications. SAS IT is a middleware that enhances SAS' ability to access and deliver data to the user. It provides a collection of API's that enable access to SAS using industry standard platforms, namely Java 2 Platform Enterprise Edition (J2EE), and the Microsoft Component Object model (COM), and Distributed Component Object Model (DCOM).

This paper focuses on developing Windows applications utilising the Local COM only, although an overview of DCOM will be given.

SAS has offered open client access to integrate SAS software with other applications since version 6.0, using the Object OLE Automation interfaces. In contrast SAS IT utilises the Integrated Object Model (IOM), for which there are two distinct advantages when developing Windows applications within the SAS system:

- Integration Technologies provides a hierarchy of interfaces, where as the OLE Automation interface offers only one
- Applications that use Integration Technologies objects can run SAS programs asynchronously

This paper is split into two main sections: SAS Integration Technologies Features, and Developing Windows Clients using VBA.

INTEGRATION TECHNOLOGIES FEATURES

SAS IT integrates SAS with industry standard tools written in a variety of languages to create applications, and provides the functionality of many middleware technology to enable better communication between SAS and other applications.

SAS Integration Technologies combines the following features:

- Directory Services

- Messaging Middleware
- Distributed Objects
- Integrated Object Model (IOM)

In this paper we will focus on IOM, although an overview of the other three features will be given.

DIRECTORY SERVICES

Directory services technology enables you to collect information about users, applications, files, access control, and other resources, and store this information into a common directory. This directory is accessible to all users and directory enabled applications on the network, and is not specific to any one application. This is useful as it means all information can be easily managed, as any changes are available to all users and directory enabled applications.

Directory services technology incorporates Lightweight Directory Access Protocol (LDAP) into your SAS programs. LDAP is an industry standard protocol for accessing online directory services. Directories such as Microsoft Active Directory, and Sun ONE are supported. SAS call routines can be read as well as write to metadata in/from an Enterprise Directory Server. If you are using the SAS information Service, there is a JAVA API which allows easy access to metadata stored in LDAP repositories from a Java program.

MESSAGING MIDDLEWARE

SAS IT supports IBM's WebSphere MQSeries, Microsoft's Message Queuing Services (MSMQ) and Tibco TIB/Rendezvous messaging queuing software. These allow implementation of asynchronous communications.

Requests are sent, and responses are received, as 'messages'. SAS provides an interface to these products, with this interface, SAS programs can create new MSMQ message queues or take advantage of existing ones.

Use of Message Queuing will not be covered in this paper.

DISTRIBUTED OBJECTS

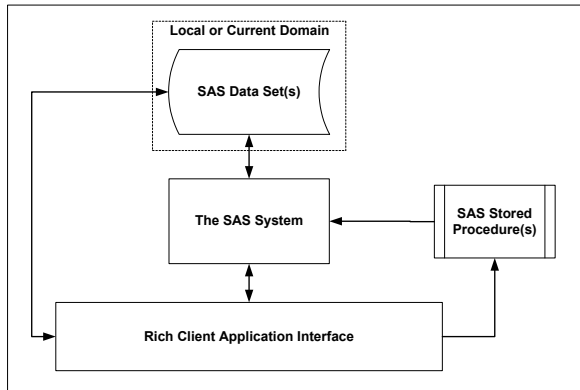
Windows clients written in languages such as Visual Basic or Visual C++, access IOM servers running in the Windows operating environment using Microsoft's Component Object Model (COM/DCOM). COM is a language-neutral binary interface specification for Windows objects and contains a set of run-time functions for instantiating them. With COM, it is possible to build an object using one language, and use that object in another language.

If you want to use remote sessions you need to use the Distributed Component Object Model (DCOM) interface. This enables you to access SAS on a server without having SAS licensed on the local machine. To be able to do this you need to have SAS Integration Technologies licensed.

A local COM connection is established locally, meaning SAS is on the same machine as your data, and the application that is accessing SAS. Figure (1) shows how the Client application can access SAS, SAS Data Set(s) and SAS Stored Process (to be discussed later).

Both local COM and DCOM provide a way by which two objects in different object spaces or networks, can talk together by calling each others methods. They allow us to access the SAS engine, such as the data step, proc step and libraries.

Figure 1: Local COM

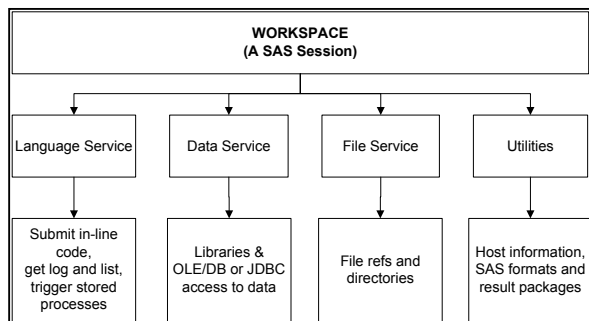


INTERGRATED OBJECT MODEL(IOM)

IOM is a programming model in SAS IT that provides distributed object interfaces to Base SAS software. The IOM interfaces are available with Base SAS on Windows, however their use is restricted to local COM unless you have licensed SAS Integration Technologies. The IT licence grants you the right to make IOM calls through DCOM and TCP/IP. Windows clients accessing IOM servers on non-Windows server platforms use the IOM Bridge for COM.

IOM provides classes and methods that can be used to allow communication between the front-end and SAS. Objects are integrated firstly as part of the SAS system and secondly, as part of the application.

Figure 2: Integrated Object Model



Language Service: - provides an interface to submit SAS code, and contains the following features:

- Allows code from the client to be submitted
- Can return the log file
- Can return the output file
- Allows events to be raised on:
 - ERROR in the submitted code
 - DATA step starts and stops
 - PROC step starts and stops
 - Submitted code completes
 - Stored Processes can be called

Data Service:- Allows SAS libraries to be created, removed and browsed in the current work space. Developers may

opt to use the IOM Provider instead of accessing the SAS data sets directly through the Data Service Interface.

File Service Interface:- Allows files and file references to be created, removed and browsed from the current SAS workspace. There is an additional text stream object for reading ASCII or EBCDIC text files.

Utility Service Interface:- Allows SAS options to be set and retrieved (equivalent to the OPTIONS statement). It does not seem to allow command line options to be passed. Allows the operating system version to be returned on which the SAS Workspace is running. Options which can only be set at SAS launch cannot be controlled.

SAS WORKSPACE MANAGER

Figure 2 uses the term Workspace, the workspace is the object at the root of the IOM hierarchy. A SAS Workspace is similar to a Display Manager in SAS, a good way to consider a workspace is as an embedded SAS session running in the application you are developing. This workspace has several attributes, properties and methods which utilise the power of the SAS System. All SAS Workspaces are managed by the Workspace Manager.

The main job of the Workspace Manager is to establish and manage connections with SAS Workspaces. A workspace is created via the Workspace Manager, an ActiveX control that performs three functions:

- establishes a connection with SAS – returning a workspace
- enables selection between running SAS workspaces, so you can obtain previously created SAS Workspaces
- provides access to a workspace from with web pages

Lets look at a VBA program and create a workspace, to begin with you will need to create a workspace. There are two ways to use the Workspace Manager to establish a connection to SAS:

Unadministered, (standalone)

In this case all of the connection information for the workspace is hard coded (where all connection information is provided in the VBA code). Below is an example of code that could be used to create an instance of a SAS session on a local machine:

```

'Declare the VB variables to be used
Dim Ws As New
SASWorkspaceManager.WorkspaceManager
Dim obSAS As SAS.Workspace
Dim Info As String
'Set the workspace
Set obSAS =
Ws.Workspaces.CreateWorkspaceByServer ("",
VisibilityProcess,Nothing "", "", "info")
  
```

One reason for using an unadministered connection is that we can specify the "visibility" of the server.

There are two settings:

- **VisibilityNone:** which means the WorkspaceManager will not keep track of the created workspace
- **VisibilityProcess:** which specifies that all calls within the same process can access the created workspace

The visibility is set to VisibilityProcess which will allow an ADO connection to the Workspace. The info parameter is a string

that contains details about the server, used to establish a connection; this is optional.

Administered, (networked)

Where all connections information is provided in either a file or LDAP. It is at this point we indicate which machine to connect too.

To create an administered workspace use the following:

```
'Declare the VB variables to be used
Dim Ws As New
SASWorkspaceManager.WorkspaceManager
Dim obSAS As SAS.Workspace

'Set the workspace
Set obSAS =
Ws.Workspaces.CreateWorkspaceByLogicalName
("CAMP", VisibilityProcess, "", "", "")
```

A useful feature of using an administered workspace is you can more easily change the machine where SAS is running.

DEVELOPING WINDOWS APPLICATIONS

There are several tools that can be used by programmers when building applications, we are going to look at a simple application using local COM and developed in Excel, using VBA and ADO to manipulate the data.

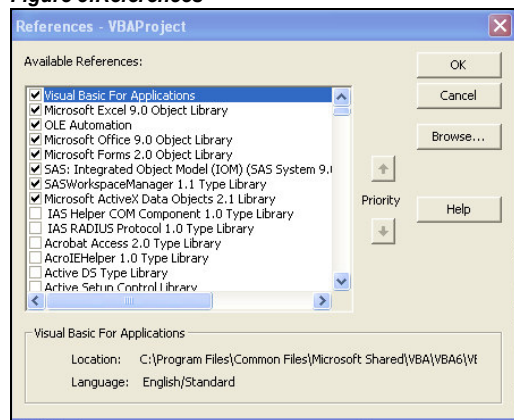
All coded examples used in this paper have been run using SAS V8.2, and Microsoft Excel 2000 version 9.0. It is not the intention of this paper to explain ADO, or VBA.

Requirements:

- Base SAS
- Microsoft Excel, where we need to reference ADO, SAS IOM and SAS Workspace Manager in VBA Design Environment, as described below:

Open Excel on the PC that the spreadsheet will run from, press Alt+F11, to open Microsoft Visual Basic. Go to the Tools drop down menu and select References. You will see the following Dialog Box appear, scroll through the options until all the necessary components have been selected as shown below.

Figure 3: References



We need to reference these DLL's so we have access to the their methods. These methods allow us to access the functionality to communicate with SAS.

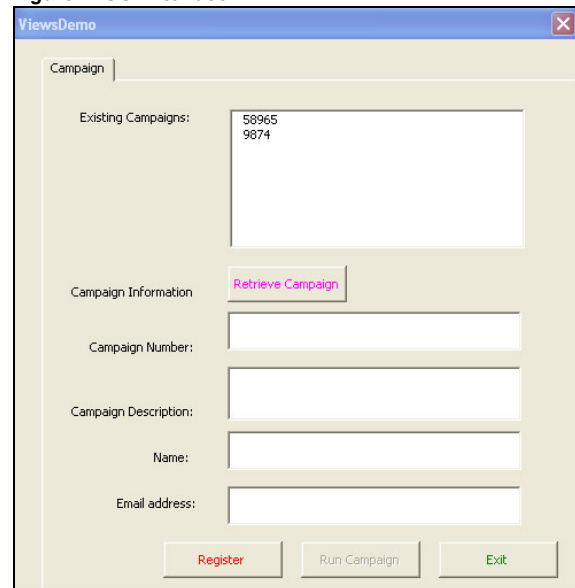
SAS recommends you use the most recent references, so

although I am running version SAS 8.2, my references (IOM) are for SAS 9.0.

The application being demonstrated is a VBA GUI interface, which allows the users to retrieve campaigns and register campaigns.

When the user opens the spreadsheet it automatically loads the current campaigns; they then select the campaign they wish to view. Figure 3 shows the GUI interface created in VBA, displaying the current campaigns.

Figure 4: GUI interface



To explain how the application works, we will step through the code and explain it's functionality. To begin with the application starts using an onload function where it creates a SAS workspace, defines the IOM provider, run's a stored process and defines a permanent SAS library.

CREATE SAS WORKSPACE

To start a SAS session we need to create an instance of the SAS Workspace. As this code is likely to be reused many times it is good practice to create a standalone function or sub routine.

First we create an instance of the SAS Workspace Manager (WS), which controls all instances of a SAS Workspace, and a SAS Workspace(obSAS). We are creating a workspace by logical name and calling it CAMP, allowing us to retrieve this workspace by the name CAMP, throughout our SAS session, it can also be referenced using an ID.

```
'Define a SAS workspace, using bylogicalname
and call it CAMP
Dim Ws As New
SASWorkspaceManager.WorkspaceManager
Dim obSAS As SAS.Workspace
Set obSAS =
Ws.Workspaces.CreateWorkspaceByLogicalName
("CAMP", VisibilityProcess, "", "", "")
```

IOM DATA PROVIDER

The SAS IOM Data Provider is an OLE DB data provider that gives access to SAS data sets that are managed by SAS Integrated Object Model (IOM) servers.

The IOM Data Provider can be accessed through the low-level OLE DB interfaces using high-level languages such as ActiveX Data Objects (ADO), using Visual C++, VBScript, Java, JScript, or Visual Basic.

These interfaces provide the client application with a great deal of flexibility in manipulating the SAS data. However, instead of coding directly with the IOM Provider interfaces, programs can be developed using ADO, which is a programming layer on top of the OLE DB provider.

If you are using the IOM Data Provider (a component of SAS Integration Technologies) you will need the Microsoft Data Access Components (MDAC) Version 2.1 or higher on each client machine. MDAC is available from the Microsoft Web site.

To establish a connection to a SAS Data Set we need to provide the name of the IOM Provider that we are using. In this case SAS.IOMprovider.1.

```
'Tell SAS which IOM provider to use
Dim Connection As New ADODB.Connection
Connection.Open
"Provider=sas.iomprovider.1;SAS Workspace
ID=" _& obSAS.UniqueIdentifier
```

We can then execute some SAS code, and retrieve our campaign numbers. The method we have chosen to use is a SAS stored process.

STORED PROCESS

A Stored Process in SAS terms is simply a .sas file which has a special comment in the header of:

***Processbody;**

Stored Processes can be thought of as macros within Visual Basic. The advantage of using Stored Processes is that the application logic is completely separate from the business logic contained within the Stored Process. Therefore, any changes to the business logic (our SAS code) do not impact the application logic.

To use a Stored Process in SAS we need to specify the Repository of Stored Processes (the directory containing the SAS files), create the LanguageService and the StoredProcessService, define the name of the SAS program that we want to run, and any required parameters (SAS Macro variables) to be submitted to the SAS program. If no parameters are required then a null value can be passed.

SAS Stored Process - VBA

```
'Create an instance of the SAS StoredProcess
service
Dim obStoredProcessService As
SAS.StoredProcessService

'Create an instance of the SAS language
service
Set obStoredProcessService =
obSAS.LanguageService.StoredProcessService

obStoredProcessService.Repository =
"file:D:\Views\CODE"
obStoredProcessService.Execute "autoexec",
"date=15April2004"
```

In the above VBA code we are running the stored process autoexec.sas from the 'd:\views\code' location, the autoexec code is shown below:

SAS Stored Process - autoexec.sas

```
%let date=;

*processbody;

Proc printto;
log="D:\views\logs\views_&date..txt" new;
run;

%put 'application started
and autoexec run';
%put 'This is a views demo';

Proc printto;
run;
```

The %let date statement allows parameters and values to be passed to the SAS program and to be used as a macro variable. In our program we are passing the parameter date with the value 15April2004, this is being used in the creation of the SAS log name. The parameters must have the same name, in our case date.

DEFINE A PERMANENT SAS LIBRARY

Using the DataService class and the AssignLibref method we can assign a permanent SAS library. We must pass a library name (META) and the location 'd:\views\meta.' This allows us to use the SAS data sets within this library as can be seen in the read from a data set section below.

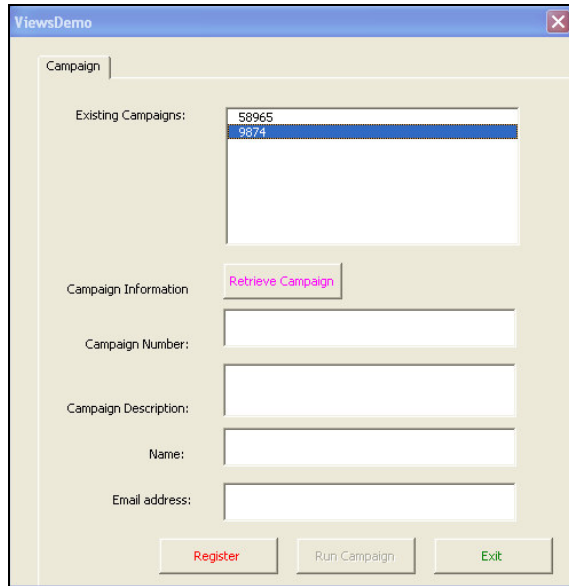
Define a permanent SAS library within the workspace CAMP:

```
'Declare SAS Workspace Objects
Dim Libref As SAS.Libref
Set obSAS =
Ws.Workspaces.GetWorkspaceByName ("CAMP")
Set Libref =
obSAS.DataService.AssignLibref("META", "",
"D:\VIEWS\Metadata", "")
```

READ FROM A DATA SET

We now are going to read from a data set; to do this we are going to use ADO and a recordset object. The user makes a campaign code selected by highlighting the campaign number and then clicks the Retrieve Campaign Button, where upon the data is retrieved from the SAS Data Set.

Figure 5: Retrieve Data



```
'Open the recordset that was
previous defined
obRecordset.Open "meta.camp", Connection,
adOpenDynamic, adLockPessimistic,
ADODB.adCmdTableDirect

'Apply a filter (similar to a SAS where
clause)
obRecordset.Filter = "camp_id = '" & camp &
"'"

Camp_No = obRecordset.Fields("Camp_id")
Camp_desc = obRecordset.Fields("Camp_desc")
Email_Name =
obRecordset.Fields("Email_Name")
Email = obRecordset.Fields("Email")

'Close the recordset
obRecordset.Close

Register.Enabled = False
Camp_No.Enabled = False
Run.Enabled = True

End If
```

Retrieve Campaign Code -VBA

```
'Declare ADO Connection & Recordset Objects
Dim obRecordset As New ADODB.Recordset
Dim rs As New ADODB.Recordset

'Open the recordset object
obRecordset.Open "META.CAMPAIGN",
Connection, adOpenDynamic,
adLockPessimistic, ADODB.adCmdTableDirect

While Not (obRecordset.EOF Or
obRecordset.EOF)

ViewsDemo.GetCamp.AddItem
obRecordset.Fields("CAMPAIGN_ID")

obRecordset.MoveNext
Wend

'Close the recordset object
obRecordset.Close
```

We have moved through the SAS data set one record at a time retrieving the campaign number and have displayed this to the screen in the GETCAMP object, which is labelled existing campaigns on the GUI interface.

We are then going to click the retrieve button and retrieve the campaign number 9874, once we click on this button the following code is executed:

```
Dim camp As String

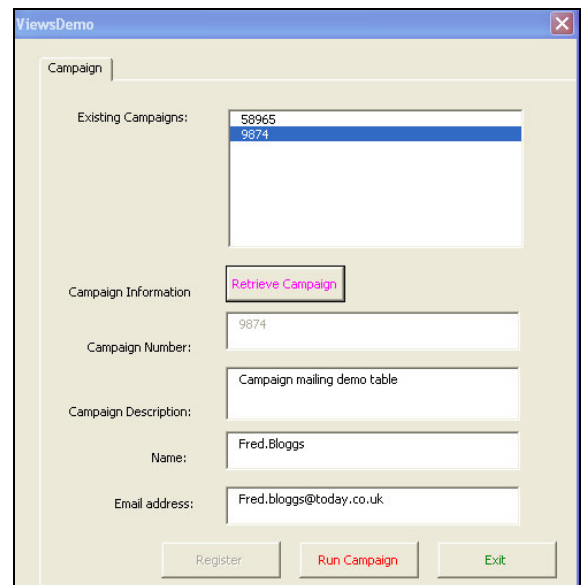
If GetCamp.Value = "" Then

MsgBox "Please Select a Campaign Number",
vbOKOnly, "Retrieve Campaign"

Else

camp = GetCamp.Value
```

Figure 6: Retrieved campaign information



The information has been retrieved and displayed to the screen, the user can now run the campaign and exit. This simple demonstration will hopefully get you thinking about what you can do.

CONCLUSION

By introducing SAS Integration Technologies, SAS have provided us with the mechanism to utilise the power and functionality of SAS Software as the back end to User Interfaces, which can be written in a variety of industry standard languages. This paper introduced some of the key functions that a developer may wish to incorporate into their applications.

ACKNOWLEDGMENTS



Developing an Open Client in Visual Basic, Daniel Jahn, SAS Institute

<http://support.sas.com/rnd/itech/papers/VBClient.pdf>

SAS Integration Technologies Library

<http://www.sas.com/rnd/itech/library/index.html>

<http://support.sas.com/onlinedoc>

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Contact the author at:

Author Name	Lynn Pope
Company:	Amadeus Software Limited
Address:	Orchard Farm, Witney Lane, Leaffield, Oxon OX29 9PG
Work Phone:	01993 878287
Fax:	01993 878042
Email:	lynn.pope@amadeus.co.uk
Web:	www.amadeus.co.uk

SAS and all other SAS Institute Inc. product or service names are registered trademarks 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.