USING SAS INTERGRATION TECHNOLOGIES TO INTERFACE WITH ENTERPRISE APPLICATIONS WRITTEN IN VISUAL C++

Joseph Nipko, Qualex Consulting Services, Scottsdale, AZ

ABSTRACT

Utilizing SAS as an enterprise IT solution in a distributed Microsoft Windows environment centered on COM objects written in C++ can be a challenge for the best of SAS developers. SAS Integration Technologies exposes a set of COM interfaces that make integrating SAS with other custom windows applications a relatively pain free experience. This paper explores the SAS COM Object Model and demonstrates how to create an integrated application using the objects in Visual C++. Topics include instantiating and using the SASWorkspaceManager object, writing custom event handlers for IOM server runtime events, and using ADO with the IOM OLEDB Provider to access SAS data sets.

INTRODUCTION

SAS Integration Technologies (IT) provides the foundation that enables open client access to SAS software. With SAS IT developers can easily integrate SAS with industry standard tools to create powerful applications. SAS IT is middleware that greatly enhances SAS' ability to access and present data to the user. There are four main aspects to SAS IT including LDAP directory integration, an information delivery framework known as Publication/Subscribe, Message Queuing and an Integrated Object Model (IOM). In this paper we will focus on the last aspect of SAS IT, the IOM component and show how we can use this component to easily create windows programs using Visual C++.

SAS has offered open client access to integrate SAS software with other application since version 6.0 with OLE Automation interfaces. There are two important advantages to using the IOM over the Object OLE Automation interfaces in the SAS system for windows application development, namely:

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

The objects that are part of the IOM are designed mainly to submit SAS Language statements to a SAS server and get output back from SAS. A custom Windows application might use the SAS software to execute of a sequence of data extraction, summarization, analysis, and presentation steps. The result of these steps may be a collection of data sets, graphs, and formatted. SAS IT provides an easy route to executing the SAS programs and then accessing the resulting output.

The IOM interfaces are available with Base SAS on Windows, however their use is restricted to local COM unless you have licensed Integration Technologies. The IT license 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. This bridge allows you to develop native COM/DCOM applications that access server data, for example, on UNIX and mainframe platforms. This transparency is a key feature of SAS Integration Technologies. It enables application developers to have full access to the architectural elements available in the Windows environment, even when their clients communicate with servers in other operating environments.

WORKSPACE

The workspace is the object at the root of the IOM hierarchy. From this object the application can create:

- Data Service Interface: Provides access to SAS Library information and contains interfaces for reading and writing SAS datasets. Developers may opt to use the IOM Provider instead of accessing the SAS datasets directly through the Data Service Interface.
- File Service Interface: Provides an interface to read and write files and enumerate filerefs defined to SAS.
- Language Service Interface: Provides an interface to submit SAS code. Statements can be passed directly to the Submit function on the interface or code may reside as a stored process on the server callable by name.
- Utility Service Interface: Provides interfaces to set formats, options, and other utility features of a SAS session.

The entire IOM hierarchy is shown in Figure 1. These interface definitions are contained in the type library sas.tlb, consequently to use the IOM objects you'll need to import this library. Let's take a look at a Visual C++ program that connects to an IOM server, submits a few SAS statements to the server, and displays the results.

The following code can be used to start a SAS session:

```
#import "sas.tlb";
…
HRESULT hr;
SAS::IWorkspacePtr pIWorkspace;

hr = pIWorkspace.CreateInstance(
        "SAS.Workspace.1");
```

Using the newly created Workspace we can submit some SAS code, display the results in the Visual C++ debug window and then destroy the SAS server when the we are finished:

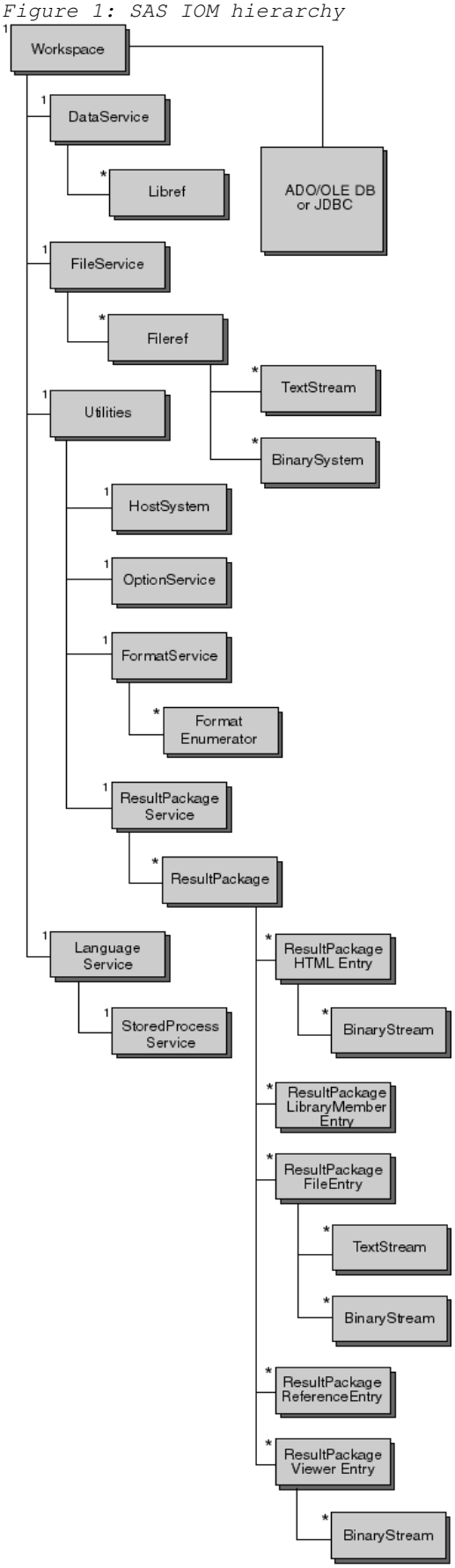
```
pIWorkspace->LanguageService->Submit(
     "proc sort data=sashelp.prdsale;\
      by year quarter;\
      run;\
      proc means data=sashelp.prdsale;\
      var actual predict;\
      by year quarter;\
      run;");

OutputDebugString(pIWorkspace->
LanguageService->FlushList(10000));

OutputDebugString(pIWorkspace->
LanguageService->FlushLog(10000));

pIWorkspace->Close();
```

Figure 1: SAS IOM hierarchy



In the above example we have only used a few of the objects available to us through the IOM. In an enterprise application we would use many of the other features in the other objects in the hierarchy.

IOM Provider

The IOM Provider delivers access to the SAS data sets that are defined to the IOM server. The SAS website contains a detailed explanation of the available interfaces for the OLEDB provider. 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 OLEDB provider. The architecture and functions available with ADO may be more familiar to the Visual Basic developer. To demonstrate using the IOM Provider with ADO the following example will create an ADO connection bind the connection to the SAS workspace, and create an ADO record set from the SAS dataset:

```
_bstr_t strProvider (
     "Provider=SAS.IOMProvider.1; ");

_bstr_t strWorkspace ("SAS Workspace ID="
     + pIWorkspace->UniqueIdentifier);

_bstr_t strConnect = strProvider +
strWorkspace;

_bstr_t strSource (
     "select * from sashelp.prdsale");

ADODB::_ConnectionPtr obConnection;
ADODB::_RecordsetPtr obRecords;

hr = obConnection.CreateInstance(
      __uuidof( ADODB::Connection  ));

hr = obConnection->Open(strConnect,
_bstr_t(""),
       _bstr_t(""), NULL);

hr = obRecords.CreateInstance(
       __uuidof( ADODB::Recordset  ));

obRecords->Open(
     strSource,
     _variant_t((IDispatch*)obConnection),
     ADODB::adOpenForwardOnly,
     ADODB::adLockReadOnly,
     ADODB::adCmdText );
```

Events

The COM events in the IOM server have been implemented in the standard connection points mechanism wherein the client application implements an event interface described by the server. Table 1 gives the event interfaces that are available in the IOM and the events that they expose:

In writing a Visual C++ application that can capture these events there are two main steps involved. First write a client event class that implements the code for each desired event type. For example, if we would like the client application to respond to LanguageService events we implement a class derived from the CILanguageService with the event handlers overwritten.

Table 1: IOM event interfaces.

Event Interface	Events
CIODSEvents	Hierarchy
	HierarchyClose
	Directory
	DirectoryClose
	HTMLFileOpen
	HTMLAnchor
	HTMLFileClose
	Output
	HTMLText
	HTMLMark
CILanguageEvents	ProcStart
	SubmitComplete
	ProcComplete
	DatastepStart
	DatastepComplete
	StepError
CIDataServiceEvents	LibraryAssign
	LibraryDeassign
CIDataServiceMemEvents	MemberCreate
	MemberDelete
	MemberReplace
	MemberRename
CILibrefEvents	MemberCreate
	MemberDelete
	MemberRename
	MemberReplace
CIODSFileEvents	FileOpen
	FileClose
	DirectoryBegin
	DirectoryEnd
	OutputElement
	AnchorElement

Below is the code for the header file for a LanguageService event handler called CLanguageEventDispatch:

```
class CLanguageEventDispatch
   :public CComObjectRoot,
    public SAS::CILanguageEvents
{ public:

CLanguageEventDispatch();
virtual ~CLanguageEventDispatch(;

BEGIN_COM_MAP(CLanguageEventDispatch)
     COM_INTERFACE_ENTRY(IUnknown)
     COM_INTERFACE_ENTRY(
                SAS::CILanguageEvents)
END_COM_MAP()

HRESULT _stdcall raw_ProcStart(BSTR Procname);
HRESULT _stdcall raw_SubmitComplete(long Sasrc);
HRESULT _stdcall raw_ProcComplete(BSTR Procname);
HRESULT _stdcall raw_DatastepStart();
HRESULT _stdcall raw_DatastepComplete();
HRESULT _stdcall raw_StepError();

};
```

An implementation for each of the above functions will also be needed. In the above header file the BEGIN_COM_MAP, END_COM_MAP, and COM_INTERFACE_ENTRY macros are needed to enter the interfaces into the application COM map so that they may be accessed by the QueryInterface function.

Next, create an instance of the client event interface and pass a pointer to this interface to the IOM Server by calling the Advise function. The following code accomplishes this:

```
// Declare a pointer to the client
// event class
CComObject<CLanguageEventDispatch>*
     ptrLanguageEvents;

// Create an Object of the client
// event class
CComObject<CLanguageEventDispatch>::Create
Instance(&ptrLanguageEvents);

// Get the IUnknown interface pointer
// for the active IOM server
pIWorkspace->LanguageService->
     QueryInterface( IID_IUnknown,
     (LPVOID*)&pIUnknownProvider);

// Use the IUnknown pointer from the IOM
// server and the IUnknown interface
// pointer from the client event class
// in a call to AtlAdvise to set up the
// bi-directional communication between
// the two
hr = AtlAdvise(
     pIUnknownProvider,
     ptrLanguageEvents->GetUnknown(),
     __uuidof(SAS::CILanguageEvents),
     &pdw);
```

After the above code is executed the client application will receive IOM LanguagueService events. If the application would like to stop receiving the events, simply call the unadvised function, as follows:

```
HRESULT hr;

hr = AtlUnadvise(
     pIUnknownProvider,
     __uuidof(SAS::CILanguageEvents),
     pdw);
```

WORKSPACEMANAGER

The main job of the WorkspaceManager is to establish and manage connections with SAS Workspaces. It can also provide pooling of the interfaces and facilitate ADO connections to active SAS sessions using the IOM OLEDB Provider. The WorkspaceManager component is implemented as a COM Singleton object; consequently, there will be a single instance of the WorkspaceManager class in any given process. This design provides a great deal of flexibility in the development of custom Windows applications. It allows the WorkspaceManager to provide pooling services since multiple workspaces can be used on different threads. Similarly, a single SAS IOM Workspace can be shared between multiple threads within the same process.

There are two ways to use the WorkspaceManager to establish a connection to SAS:

Unadministered, standalone

In this case all of the connection information for the workspace is hard coded. Below is an example of code

that could be used to create an instance of a SAS on a local machine:

```
SASWorkspaceManager::IWorkspaceManager2Ptr
pIWorkspaceManager;
BSTR xmlInfo;
SAS::IWorkspacePtr pIWorkspace;
HRESULT hr;

hr =
pIWorkspaceManager.CreateInstance("SASWorkspaceMa
nager.WorkspaceManager.1");

pIWorkspace=pIWorkspaceManager->
    Workspaces->CreateWorkspaceByServer(

// Name of the workspace to be created
_bstr_t("localsas"),

// Visibility of the created workspace
SASWorkspaceManager::VisibilityProcess,

// pointer to a ServerDef object that is
// used to set the connection information
NULL,

// Login name to run the SAS server under
// ignored for local SAS connections
_bstr_t(""),

// Password for Login Name above
_bstr_t(""),


// XML table that describes the connection
// attempts
&xmlInfo);
```

One reason for using an unadministered connection is that we can specify the "visibility" of the server. There are two settings: VisibiliyNone which means the WorkspaceManager will not keep track of the created workspace and VisibilityProcess which specifies that all calls within the same process can access the created workspace. The visibility set to VisibilityProcess will allow an ADO connection to the Workspace.

Administered, networked

This allows launching of SAS Workspaces from a shared repository (either in a file or LDAP server). SAS IT ships with the Integration Technologies Administrator, which can create the connection information on an LDAP server. To establish the connection to a remote SAS machine where the connection information has been administered into LDAP, use the same variables as in the unadministered case but make the following function call:

```
pIWorkspace=pIWorkspaceManager->
    Workspaces-> CreateWorkspaceByLogicalName(

// name of the workspace can be used to
// access the workspace

_bstr_t ("workspacename"),

// visibility of the process

SASWorkspaceManager::VisibilityNone,

// logical name used to lookup the
// connection information

_bstr_t( "LogicalName" ),

//  ReferenceDN used to give login
//  information if the COM Bridge
//  is used

_bstr_t(""),

// Describes the connection attempts

&xmlInfo);
```

Workspace Pooling

Workspace pooling allows the developer to create a pool of connections to IOM servers that can be shared among several client processes. If the client processes make many brief connections to the IOM server, pooling can help reduce connection times. There are two different mechanisms for workspace pooling that are supported by the WorkspaceManager:

- Integration Technologies Pooling: In this type of pooling the pooled workspaces are configured by LDAP, an LDIF-formatted file or by hard coding the server parameters in the code. For a detailed account of this type of pooling see the SAS IT website.
- COM+ Pooling: This type of pooling is only available on platforms that support COM+, such as Windows 2000. There are two options for configuring this type of pooling, either as a Library application in which each process has its own pool or as a Server application where the pool is shared by all the processes on the same machine.

A COM+ pool is administered using the Microsoft Management Console (MMC) that is distributed with Windows 2000. Setting up the pooled workspace using the MMC is a straight forward task facilitated by the wizards available in the MMC. The SAS IT web site gives a step by step explanation of the procedure.

Once the PooledWorkspace is registered with COM+ the following code can be used to get a pooled workspace:

```
// Set the COM/DCOM security
hr = CoInitializeSecurity(
NULL,-1,NULL,NULL,
RPC_C_AUTHN_LEVEL_NONE,
RPC_C_IMP_LEVEL_IMPERSONATE,
NULL,0x0,NULL);

SASWorkspaceManager::IPooledWorkspacePtr
pIPooledWorkspace;

// Create an instance of the pooled
// workspace
hr = pIPooledWorkspace.CreateInstance(
"SASWorkspaceManager.PooledWorkspace");

//
pIWorkspace = pIPooledWorkspace->
                    GetWorkspace();
```

You must call the CoinitializeSecurity function to set the COM security level before calling CreateInstance. In the above section of code I've set the DCOM security level to NONE for the application. You can manage the security settings for DCOM through the use of the DCOMCNFG utility which is detailed in a Microsoft FAQ on COM security. The pooled workspace can be used, for example, to submit SAS statements through the LanguageService. However if the programmer desires an ADO connection to the SAS server through the IOM Provider then we need to have the Workspace managed by the WorkspaceManager with the visibility set to VisibilityProcess. The

AddExternalWorkspace function in the Workspaces collection of the WorkspaceManager can be used for this purpose as follows:

```
hr = pIWorkspaceManager->Workspaces->
AddExternalWorkspace(

//  Set the visibility of the
//  added workspace
SASWorkspaceManager::VisibilityProcess,

//  pass the pooled workspace pointer
pIWorkspace);
```

CONCLUSION

The introduction of the open client architecture supplied by SAS through Integration Technologies makes it easy to create powerful custom Windows applications. With SAS IT companies already using SAS as an analytic tool can quickly and easily use this technology to integrate SAS programs into custom enterprise applications.

REFERENCES

SAS Integration Technologies Library

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

SAS Integration Technologies Overview White Paper

Steve Jenisch

Microsoft COM Security FAQ:

http://support.microsoft.com/support/kb/articles/Q158/5/08.asp

CONTACT INFORMATION

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

Joseph Nipko
Qualex Consulting Services, Inc
15947 N 102nd Place
Scottsdale, AZ 85255
Work Phone: 602-410-4552
Fax: 815-550-5182
E-mail Address: joe@qlx.com