Paper 003-30

Through the Looking Glass: Two Windows into SAS®

Peter Eberhardt, Fernwood Consulting Group Inc., Toronto, ON
Richard A. DeVenezia, Consultant, Remsen, NY

ABSTRACT

How do you like your SAS? With a cup of Java or scooped by a .NET? It has always been true that with SAS there were many ways to solve your problems, but when it came to visual interface your options were limited. That changed in v8 with the introduction of SAS/Integration Technologies and the Integrated Object Model (IOM); SAS can now be seen through a myriad of windows. In this paper we will look at one problem and the SAS code to address it. The fun part is that we will show how this common SAS code can be accessed through two different environments – Java and Microsoft .NET.

INTRODUCTION

SAS code – the data step and procedures – can be combined in any number of imaginative ways to help solve the problems faced by organizations large and small. A perusal of any SUGI proceedings will show, there is virtually no limit on the imaginative ways SAS has been put to work. Unfortunately, the visual interface did not offer the same level of flexibility and imagination, as did the working code. With SAS v8 came the introduction of SAS/Integration Technologies (IT), and the core foundation of IT, the Integrated Object Model (IOM). The IOM opened a window to the world of SAS from virtually any programming environment. The IOM allows us to separate the interface from the engine. This paper will show how common SAS code – the engine -can be accessed through two different programming environments, Java and C#.NET. But before we jump into the code, lets review the IOM.

THE INTEGRATED OBJECT MODEL

The IOM is the heart of Integration Technologies ability to interface with multiple and diverse programming environments; it is a deceptively simple object model that exposed all of the power of SAS to your programming environment. The following figure shows the IOM hierarchy:

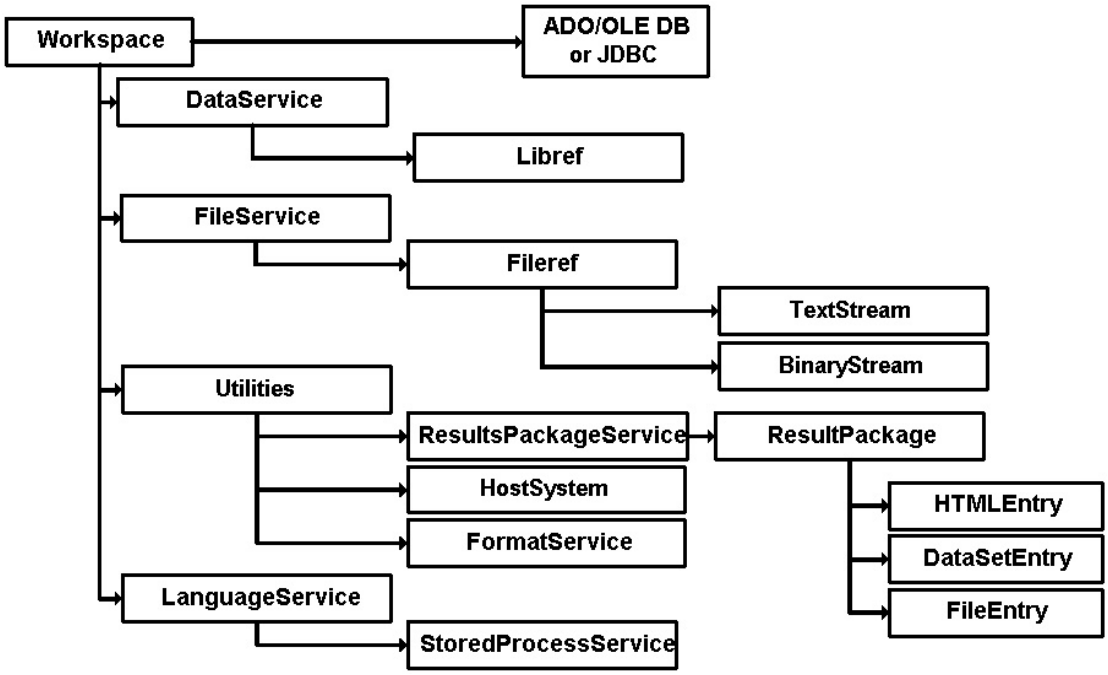

Figure 1. **The IOM Hierarchy**

The root of the IOM hierarchy is the SAS Workspace object; once instantiated within a client program, the SAS Workspace object can be thought of as a SAS session. Virtually all of the functionality you would have in a batch SAS session is available to you through the workspace object. There are two ways to create a Workspace object – using the Object Manager or using the Workspace Manager. The Workspace Manager was the only way to create a workspace in v8; although it is still available in v9, SAS suggests you use the Object Manager to instantiate and manage your IOM Workspaces to take advantage of new features. In either case, both create the SAS Workspace object on an IOM Server. In the Windows environment there are three ways the Workspace can be instantiated:

- Through local COM if the SAS Server runs on the same machine as the client
- Through DCOM if the SAS Server runs on another machine that supports DCOM
- Through the IOM Bridge for COM if the SAS Server runs on another machine that does not support COM/DCOM functionality (Unix/OS390)

Regardless of how the SAS Workspace is created (COM, DCOM, IOM Bridge), it still offers the same set of services – DataService, FileService, LanguageService, and Utilities.

LANGUAGESERVICE

The LanguageService component provides methods to submit SAS code to the IOM Server as well as retrieve log and list outputs. In addition to submitting SAS code, the LanguageService allows you to execute SAS Stored Processes – a special format of a SAS program available on the server. Using SAS Stored Processes allows you to have centralized SAS programs that can be invoked by a multitude of SAS clients.

Beyond executing SAS code, Stored Processes can also return ResultPackages. A ResultPackage is a file that can contain heterogeneous content – SAS data sets, ODS output, external files, etc. Once returned from the execution of the Stored Process, the ResultPackage can be rendered using the Utilities Service.

In order to monitor the progress of your SAS Stored Process, the IOM will raise events that the LanguageService can trap; common events such as DATA STEP or PROC begin and DATA STEP and PROC end events can be raised and trapped. Moreover, errors in the SAS execution can raise events that can also be trapped through the LanguageService.

DATASERVICE

The DataService is used primarily to manage SAS librefs – references to SAS libraries, be they SAS libraries or external (e.g. ODBC, ORACLE) libraries. You can assign, deassign or iterate through SAS librefs. All librefs managed by the IOM are relative to the server upon which the workspace is running.

FILESERVICE

The FileService is used primarily to manage SAS filerefs, external file references. The FileService can not only assign and deassign SAS filerefs, it can also be used to read and/or write to these external files. All filerefs managed by the IOM are relative to the server upon which the workspace is running.

UTILITIES

The Utilities service provides methods to access much of the non-core functionality of SAS. Two of the most useful component of the Utilities service are the FormatService and the ResultPackageService. The FormatServices allows you to apply SAS formats to the data returned using the data access component, while the ResultPackageService allows you to render the packages returned from a Stored Process.

DATA COMPONENT

The SAS IOM works with Microsoft's ADO/OLE DB, ADO.NET or a JDBC data model to share data between the client application and the SAS IOM server. The ADO/OLE DB model will work in any Windows environment, while the ADO.NET will only work in a .NET environment. The simpler ActiveX Data Object (ADO) model is shown in Figure 2.

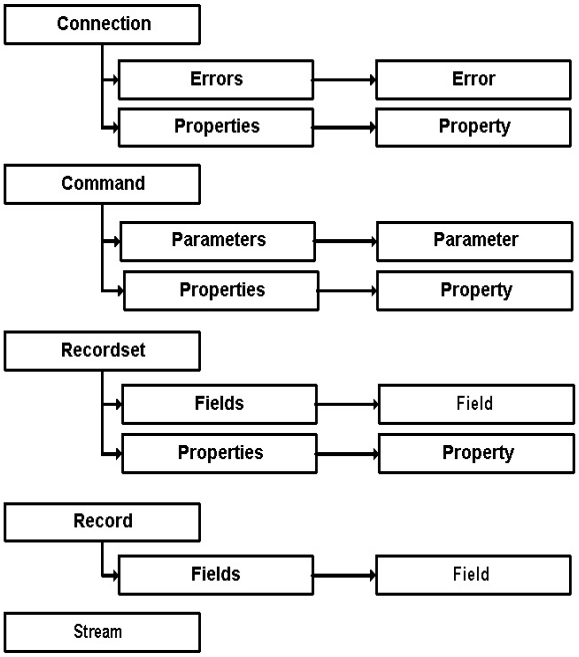

Figure 2. **The ADO Hierarchy**

MICROSOFT ADO AND ADO.NET

In order to share data between SAS and our Windows client application we will need to understand a bit about ADO and/or ADO.NET. ADO is the simpler and more mature model. As can be seen from the above hierarchy, ADO is a relatively simple and easy to use architecture. For most IOM purposes there are two objects of interest – the Connection object and the Recordset object.

The Connection object provides properties to define the source of the data, and methods to manage the link between the clients to the datasource. The Recordset object uses the Connection object to return data to the client. The Fields collection of the Recordset object provides data about the contents of the recordset.

ADO.NET provides a richer but more complex interface. The C# solution presented here uses the ADO.NET data component. Again, there are Connection and Command objects, similar to the ADO Connection and Command objects, but the Recordset object has, in essence, been split into multiple objects. The main claim to fame for ADO.NET over ADO is the ability for ADO.NET to scale.

THE PROBLEM

The business problem we are going to model is the display of baseball statistics. Our purpose is to demonstrate some basic techniques of programming to the IOM, not to demonstrate user interfaces. To that end, the user interface is very simple. And as example code, it highlights the main concepts without the rigorous care for error trapping that a production application would take.

COMMON TASKS

Our sample solutions will show:

- how to create an IOM server
- how to submit SAS code
- how to retrieve the SAS log
- how to retrieve a SAS dataset
- how to submit a SAS Stored Process
- how to trap events raised during the execution of a SAS stored process

COMMON TASKS CODE SAMPLES

For each common task, sample .NET will be shown followed by Java code.

CAVEAT EMPTOR

The SAS documentation for Windows programming to the IOM uses VB6 or C/C++ syntax most of the time. If you are new to the .NET environment you may need to review the SAS sample code carefully. The following C# code was written on a Win2000 Pro client using Visual Studio .NET 2003 Professional. The server gandalf was running Win 2000 server and SAS 9.13.

The Java code was developed in the NetBeans 4.1 IDE. The coding patterns of SAS/IT objects and method call sequences are the same as those described in Philip R Holland's publication "Could Enterprise Guide have been written in Java?", http://www.hollandnumerics.co.uk/pdf/EG_in_Java.pdf

CREATING A SAS WORKSPACE

.NET

The original (SAS v8) method to create a workspace was to use the Workspace Manager. Code to do this is:

```
// xmlinfo is the used to return the create message
string xmlInfo;

SASWorkspaceManager.ServerDef server = new SASWorkspaceManager.ServerDef()
;

server.DomainName = "fernwood.local";
server.MachineDNSName = "gandalf";
server.Port = 8591;
server.Protocol = SASWorkspaceManager.Protocols.ProtocolBridge;
sasWM = new SASWorkspaceManager.WorkspaceManager();
sasWS = (SAS.Workspace)sasWM.Workspaces.CreateWorkspaceByServer
   ("TestWS", SASWorkspaceManager.Visibility.VisibilityProcess,
    server, "fernwood\\sasdemo", "sasdemo", out xmlInfo);
```

This code works in both a v8 and v9 environment. If your application will be strictly v9, then you can use the SAS preferred method of using the Object Manager:

```
SASObjectManager.ObjectFactory obObjectFactory = new
SASObjectManager.ObjectFactory();

SASObjectManager.ServerDef obServer = new SASObjectManager.ServerDef();

obServer.MachineDNSName = "gandalf";
obServer.Protocol = SASObjectManager.Protocols.ProtocolBridge;
obServer.Port = 8591;

sasWS = (SAS.Workspace)obObjectFactory.CreateObjectByServer
      ("TestWS", true, obServer, "fernwood\\sasdemo", "sasdemo");
```

In both cases, note that the code is creating a workspace on the server named *gandalf*, using the account *fernwood\sasdemo* (with password *sasdemo*). To use this code in your own environment, you would need to change the DomainName, MachineDSNName, as well as the username and password on the create call.

JAVA

The host *extreme* is running objspawn.exe as a service that listens on port *5703*. A special application account was created to provide access to server side resources that the application needs. The reference to the workspace is maintained as a field of the class (this).

```
// Application has its own account on host providing IOM service
Properties connProperties = new Properties();
  connProperties.put("host", "extreme");
  connProperties.put("port", "5703");
  connProperties.put("userName", "app01");
  connProperties.put("password", "sugi30");
Properties [] connList = {connProperties};
WorkspaceFactory wFactory = new WorkspaceFactory(connList,null,null);
WorkspaceConnector connector = wFactory.getWorkspaceConnector(0L);
this.sasWorkspace = connector.getWorkspace();
```

SUBMITTING SAS CODE

.NET

To submit SAS code we need to use the LanguageService. In this code snippet we see how to get a LanguageService object (lang), and then submit the simple SAS data step (lang.SubmitLines). Note the conversion of C# sting variables to System.Array variables. Most of the IOM methods use System.Array variable to pass arguments, while C# more naturally uses string arrays.

```
SAS.LanguageService lang = sasWS.LanguageService;
string[] sasPgmLines = {"data shoes;", "set sashelp.shoes;", "run;"};
System.Array linesVar = sasPgmLines; //same as type of ref parm
lang.SubmitLines(ref linesVar);
```

JAVA

The SAS program to be submitted is being displayed in a jTextArea.

```
jTextAreaSubmit.setText("data shoes;\n  set sashelp.shoes;\nrun;");
...
ILanguageService sasLanguage = this.sasWorkspace.LanguageService();
sasLanguage.Submit(jTextAreaSubmit.getText());
```

RETRIEVING THE SAS LOG

.NET

As a diagnostic and debugging tool, the SAS log is invaluable. When programming to the IOM you can retrieve the SAS log (or the SAS listing, though it is not covered here). One important point to note: once you retrieve the SAS log (or list), it is lost unless you have saved it in your programme.

```
SAS.LanguageService lang = sasWS.LanguageService;

bool bMore = true;
while (bMore)
{   System.Array CCs;
    const int maxLines = 100;
    System.Array lineTypes;
    System.Array logLines;

    lang.FlushLogLines (maxLines,out CCs, out lineTypes, out logLines);

    for (int i=0; i<logLines.Length; i++) {
        RetrieveLog += (logLines.GetValue(i) + "\r\n");
    }
    if (logLines.Length < maxLines)
    bMore = false;
}
MessageBox.Show(this, RetrieveLog, "SAS LOG");
```

This code is behind a command button, but it could be behind an event to trap errors and show the log.

JAVA

FlushLogLines is used to retrieve and clear the SAS Log. The log lines are displayed in a JTextArea.

```
ILanguageService sasLanguage =
this.sasWorkspace.LanguageService();CarriageControlSeqHolder ccsh = new
CarriageControlSeqHolder();
LineTypeSeqHolder ltsh = new LineTypeSeqHolder();
StringSeqHolder ssh = new StringSeqHolder();

sasLanguage.FlushLogLines (Integer.MAX_VALUE,ccsh,ltsh,ssh);

for (int i=0;i<ssh.value.length;i++) {
    jTextAreaLog.append("\n"+ssh.value[i]);
}
```

RETRIEVING A SAS DATASET

.NET

This example shows how to create a library reference (libref) to a directory on the server, then how to submit SAS code to the server. After submitting the code we use ADO.NET to retrieve the created dataset. Important here is the use of the UniqueIdentifier property to tell the IOM which server has the data set.

```
SAS.LanguageService lang = sasWS.LanguageService;
SAS.Libref libref ;
libref = sasWS.DataService.AssignLibref
                       ("baseball", "", "c:\\baseball", "");
string[] sasPgmLines = {"data teams;",
                        "   set baseball.teams;",
                        "   keep yearID name w l;",
                        "run;" } ;
System.Array linesVar = sasPgmLines;
lang.SubmitLines(ref linesVar);

string id = sasWS.UniqueIdentifier;
string selCMD = "select * from work.teams";
string selSAS = "Provider=sas.IOMProvider.1;SAS Workspace ID="+id+";" ;

System.Data.OleDb.OleDbDataAdapter obAdapter = new
System.Data.OleDb.OleDbDataAdapter(selCMD, selSAS);

System.Data.DataSet obDS =  new System.Data.DataSet();

// Copy data from the adapter into the data set
obAdapter.Fill(obDS, "sasdata");
gridRetrieve.SetDataBinding(obDS, "sasdata");
```

JAVA

The DataService is used to deliver the result of a SAS query to a client which displays them in a jTable.

```
IDataService sasData = sasWorkspace.DataService();
Connection c = new MVAConnection(sasData, new Properties());
Statement s = c.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
ResultSet rs = s.executeQuery("SELECT * FROM SHOES");
ResultSetMetaData rsmd = rs.getMetaData();

Vector<String> columns = new Vector<String>();
```

```
for (int i 1;i< rsmd.getColumnCount();i++) {
  columns.addElement (rsmd.getColumnLabel(i));
}

Vector<Vector> rows = new Vector<Vector>();
while (rs.next()) {
    Vector<Object> row = new Vector<Object>();
    for (int i=1;i<=rsmd.getColumnCount();i++){
        row.addElement(rs.getObject(i));
    }
    rows.addElement(row);
}
jTable1.setModel (new DefaultTableModel(rows,columns));
```

SUBMITTING A STORED PROCESS

.NET

Stored Processes are central to programming to the IOM; they are truly the element that separates the interface from the engine. This example shows using the DataService to assign a libref to our data (remember, the libref is relative to the server), identifying the repository for the stored process (for simplicity we did not query the meta data server here). In addition to submitting the Stored Process (spBaseBall) with hard coded parameters, we also retrieved the data set that was created and attached the results to the data grid on the form.

```
SAS.Libref libref;
libref= sasWS.DataService.AssignLibref
                              ("baseball","","c:\\baseball", "");
// create the stored process server
SAS.StoredProcessService sp =
                     sasWS.LanguageService.StoredProcessService;

// tell it where to find the stored processes
sp.Repository = "file:c:\\baseball\\procs";

// and execute the stored process
sp.Execute("spBaseBall", "type=Y  year=2000");

// retrieve the data
string id = sasWS.UniqueIdentifier;
string selCMD = "select * from work.teams";
string selSAS = "Provider=sas.IOMProvider.1; SAS Workspace ID=" +id+ ";";

System.Data.OleDb.OleDbDataAdapter obAdapter = new
System.Data.OleDb.OleDbDataAdapter(selCMD, selSAS);

System.Data.DataSet obDS =  new System.Data.DataSet();
// Copy data from the adapter into the data set
obAdapter.Fill(obDS, "sasdata");
gridExecute.SetDataBinding(obDS, "sasdata");
```

JAVA

The data libray the stored process needs is assigned. The data and the stored process library are both network shares the server can access.

```
IDataService sasData = this.sasWorkspace.DataService();
sasData.AssignLibref("baseball", "", "\\\\extreme\\lahman",""); // libref
```

```
                = this.sasWorkspace.LanguageService().StoredProcessService();
sps.Repository("file:\\\\extreme\\baseball\\programs");
sps.Execute("spTeamsOfYear", "year=2000");
```

TRAPPING RAISED EVENTS

.NET

Want to know when one step ends and another starts? Need to know when an error occurred? The IOM raises events we can trap and use to monitor progress or flag errors in our SAS code. In C# we first need to create functions that will be used to deal with the event, Unlike VB6, C#.NET allows you to turn off and turn on trapping in a relative easy way. First you need to define you event handler:

```
private void logDSStart()
{ txtProgress.Text += "[LanguageService Event] DATASTEP start.\r\n"; }
```

This handler will write a line to a text box every time it is invoked - whenever a data step starts,

Next you need to attach it to the event that is being raised:

```
lang.DatastepStart +=
  new ILanguageEvents_DatastepStartEventHandler(this.logDSStart);
```

Then, as each data step starts, a message is written to the text box.

Finally, you can decide to stop capturing the event; in which case you code::

```
lang.DatastepStart -=
 new CILanguageEvents_DatastepStartEventHandler(this.logDSStart);
```

The C# process is a simple add a handler (+=) and remove a handler (-=).

JAVA

You can find information on trapping events in Java at this URL:
http://support.sas.com/rnd/itech/doc9/dev_guide/dist-obj/javaclnt/javaprog/javastub/eventcon.html

COMMON TASKS – COMPARISON OF THE TWO APPROACHES

The SAS code is the same. And the Java and C# code is very similar because the libraries of SAS/IT objects made available to each client environment are nearly identical. Programming to an interface (IOM) does require some level of consistency; however, the coding patterns dealing with Language events are quite different.

CONCLUSIONS

Programming in SAS means one problem, many solutions. In the old SAS world, the different solutions meant different SAS code to do the same thing. In the world of SAS/IT, we now have the opportunity to introduce one SAS solution and multiple user interfaces. In this paper we showed a simple problem and two different user interfaces that used the same SAS code to resolve the problem. Is one better? The answer is a definite YES! Of course, the answer to which one depends upon what tools, environment and skills you have. In a strict Windows environment, a .NET solution is probably best. In a Unix environment, Java rules. In either case, it is the SAS services that make both tick.

ABOUT THE AUTHORS

Richard DeVenezia is an independent consultant and has worked extensively with SAS products for over

ten years. His specialties include developing SAS/AF applications to support ad hoc data exploration and consolidation. He has previously presented at SUGI, NESUG and SESUG conferences.

Richard has an active interest in learning and applying new technologies. He is an active contributor on SAS-L, where he looks for new techniques and offers some of his own.

Peter Eberhardt has been using SAS as in independent consultant since the early 1980's. Peter works mainly in ad-hoc data analysis and modeling. He has previously presented at SSU2001, SESUG, NESUG and SUGI.

Peter is SAS Certified Professional V8, SAS Certified Professional V6, and SAS Certified Professional - Data Management V6. In addition his company, Fernwood Consulting Group Inc. is a SAS Alliance Partner.

CONTACT INFORMATION

Richard A. DeVenezia
9949 East Steuben Road
Remsen, NY 13438
radevenz@ix.netcom.com

Peter Eberhardt
Fernwood Consulting Group Inc.
288 Laird Drive, ON M4G 3X5
Canada
(416)429-5705
peter@fernwood.ca

RECOMMENDED READING

"Could Enterprise Guide have been written in Java?", Philip R Holland, Holland Numerics Ltd.
www.hollandnumerics.co.uk/pdf/EG_in_Java.pdf