

Integrating SAS® and Microsoft .NET for Data Analysis

Mai Nguyen, Shane Trahan, Patricia Nguyen, Jonathan Cirella

RTI International, Research Triangle Park, NC

ABSTRACT

Both the Microsoft .NET framework and SAS technologies are powerful development platforms in their own domain. The .NET framework is Microsoft's strategic initiative for server and desktop development for the next decade and includes many technologies designed to facilitate rapid development of Internet and intranet applications. On the other hand, SAS is the industry leader providing enterprise software for data analytics, data mining and business intelligence. SAS software is currently used extensively at RTI in survey research for statistical analysis, data integration and reporting. This paper will show how to leverage the strengths of each platform to create integrated solutions in both desktop and web-applications. We will present code samples to demonstrate how to accomplish the following tasks:

- Create an ADO.NET connection to the SAS server
- Select and update SAS datasets using ADO.NET objects
- Execute SAS script within a .NET program

Finally, we will provide a complete example of a .NET application that finds parameter estimates to simple regression models within specific guidelines such as determining the best explanatory variables in a model while maximizing the adjusted R² of the PROC REG procedure.

INTRODUCTION

At RTI International and many other organizations in the survey research industry, the SAS system has been widely used for sampling design, statistical analysis, data validation and reports. It is also often used in nightly batch jobs to create online reports for survey monitoring and tracking purposes. The SAS system, however, has not been the preferred tool when it comes to developing applications or web sites that require a user-friendly interface with data stored in different formats: relational database, XML, HTML... Today, these applications are often developed using the Microsoft .NET or the Java 2 platform. To bridge this gap, SAS has provided various integration technologies of which one is the Integration Object Model (IOM). However, the IOM is aimed more at software developers and is not yet so widely used or known in the SAS community. In this paper, we will present the technical details to show developers how to integrate the SAS system with the Microsoft .NET platform using the SAS IOM and we will also present a small application that tightly integrates .NET and SAS together to perform a simplified backward stepwise regression..

MICROSOFT .NET FRAMEWORK

The .NET framework is Microsoft's strategic initiative for server and desktop development for the next decade and includes many technologies designed to facilitate rapid development of Internet and intranet applications. The .NET Framework was created as the infrastructure for the new Microsoft .NET Platform providing a common environment for building, deploying and running client and server applications on a variety of devices from high-end servers to desktop/laptop computers to handheld PDA devices. The major building blocks of the .NET framework are:

- Common Language Runtime (CLR). CLR is the virtual machine that manages the execution of the .NET applications. It also provides low-level services such as automatic memory management, thread management, file I/O, security and language integration.
- Base Class Library (BCL). BCL is a library of classes available to all .NET languages for building .NET applications. The BCL classes abstract and encapsulate many common functions such as file operations, graphic rendering, database access, XML document manipulation and so forth.
- .NET Programming Languages..NET languages are computer programming languages that are used to produce programs that execute within the .NET framework. A unique capability of all .NET programming languages is their seamless integration with support for cross-language inheritance, cross-language exception handling, cross-language debugging and so forth. The most widely used .NET languages are C#, VB.NET and C++.
- Visual Studio.NET (VS.NET). Visual Studio.NET is the Microsoft's flagship software development product for developing Windows applications, web sites, web applications and web services for the .NET platform. VS.NET provides a host of productivity-improving features such as the IntelliSense, the Integrated Source-code debugger, the Drag-and-Drop user interface designer and so on.

The basic building blocks of the .NET framework are shown graphically below.

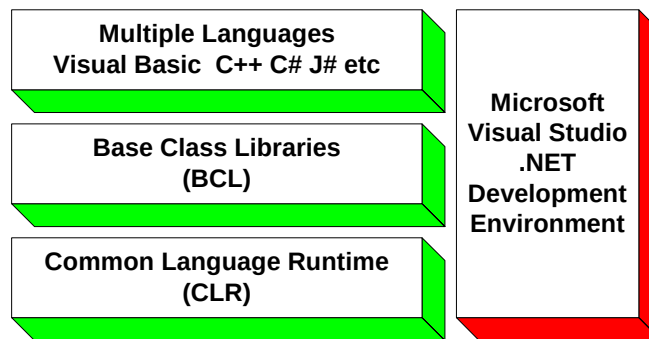
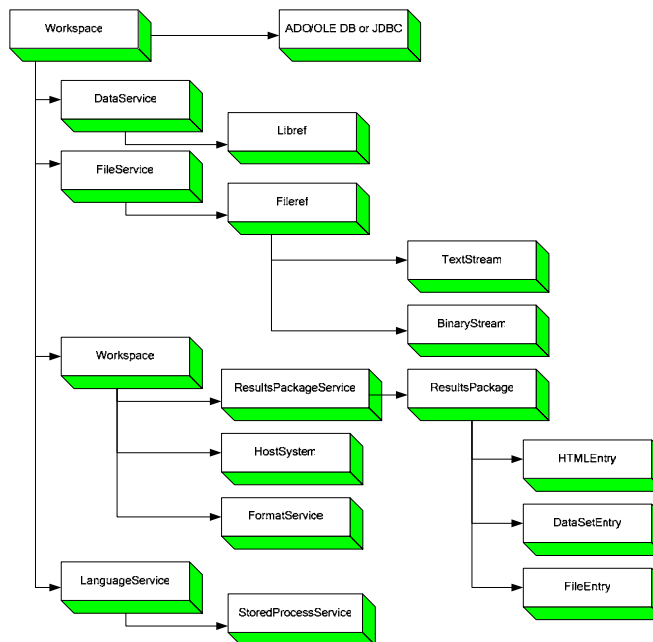


Figure 1 – The Microsoft .NET Framework

SAS® INTEGRATION OBJECT MODEL

SAS software has been used in many aspects of survey research within RTI, including sampling design, data analysis, data report and, to some extent, integration. As RTI and other organizations in the survey research industry have embraced web technologies in their survey operations in recent years, the ability to provide online data collection and to integrate data analyses and reporting becomes more essential. SAS software has always been strong on data manipulation and analysis, but it is not known for creating friendly user interfaces and data exchange, especially via the web. However, with SAS v8, SAS has introduced the SAS/Integration Technologies (IT), and its core foundation, the Integrated Object Model (IOM). The IOM opened a window into SAS from virtually any programming environment. It supported the ability to interface with Microsoft .NET as well as other major programming environments. It has a simple object model that exposed all of the power of SAS to the .NET programs. The IOM hierarchy is shown below in Figure 2.

Figure 2 - The IOM Hierarchy ¹

The usage of the SAS IOM objects will be illustrated and fully explained in our code samples.

ENVIRONMENT OVERVIEW

Although IOM is primarily designed for “thin client” applications, our code samples are running on a Microsoft Windows® PC with SAS 9 base software. The .NET sample codes are in C# using the .NET framework 1.1, though they should also work with the .NET Framework 2.0 with no modification.

We are using the standard SQL with the ADO.NET classes in the .NET framework to work with SAS datasets.

We assume the readers are familiar with the VS.NET 2003 or the latest VS.NET 2005.

¹ Source: SUGI Conference April 9-13 2005 Eberhardt DeVenezia Through the Looking Glass: Two Windows into SAS

ACCESSING SAS DATASET USING MICROSOFT ADO.NET

ADO.NET is the data access class library in the .NET base class library. It is commonly used by programmers to access and modify data stored in relational databases, though it can also be used to access data in non-relational sources. ADO.NET consists of two primary parts:

- **Data Provider.** A data provider is a set of ADO.NET classes that allow users to access a specific database, execute SQL commands and retrieve data. Essentially, it is the bridge between applications and a data source.
- **DataSet.** DataSet is an in-memory representation of any number of tables and their relationships and constraints. It allows a program to work with data retrieved from a data source in a disconnected manner.

The primary components of the ADO.NET are show below in Figure 3.

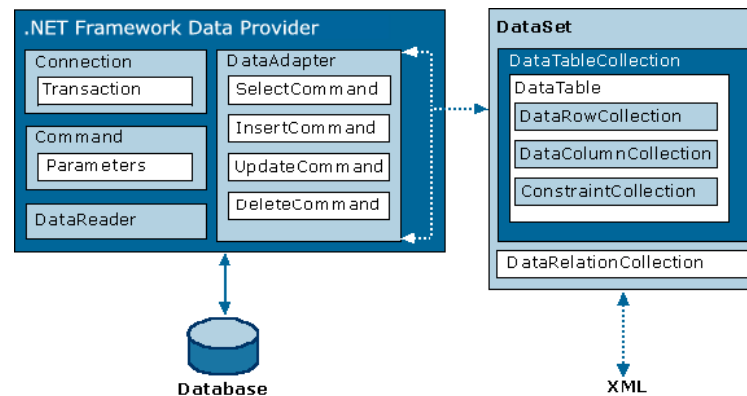


Figure 3 - ADO.NET Architecture

Our sample codes use the following steps to access data in SAS datasets using the ADO.NET objects:

- Step 1 – Add references to the SAS IOM and the SASWorkspaceManager Type Library to the project.
- Step 2 – Obtain a SAS workspace manager object from the IOM.
- Step 3 – Create an OleDb connection object using the IOM provider.
- Step 4 – Create and execute an ADO.NET command to set the location of the SAS datasets.
- Step 5 – Create an ADO.NET data adapter object to retrieve the data from SAS datasets or create and execute an ADO.NET command to modify data in a SAS dataset.

Sample code snippets for these steps are shown in the following sections.

Step 1 – Add references to the SAS IOM and the SASWorkspaceManager.

From the VS.NET Explorer Solution window, add references to following Common Object Model(COM) components:

- SAS Integrated Object Model
- SASWorkspaceManager Type Library

For convenience, the following namespace should also be added to the source files:

```
using SAS ;
using SASWorkspaceManager ;
```

Step 2 – Obtain SAS workspace manager object from the IOM

Obtain a SAS workspace manager object from the IOM using code such as the snippet below. The SAS workspace manager object will be needed to create an OleDb connection object with the IOM data provider.

```
// Create SAS workspace
SAS.Workspace sasWS ;
string xmlInfo ;
```

AD11

```
WorkspaceManager sasWM = new SASWorkspaceManager.WorkspaceManager() ;  
sasWS = (SAS.Workspace)sasWM.Workspaces.CreateWorkspaceByServer(  
    "LocalWS", Visibility.VisibilityProcess, null, "", "", out xmlInfo) ;
```

Step 3 – Create an OleDbConnection object

The following code snippet shows how to create an OleDbConnection object using the IOM data provider.

```
// Create a new OleDbConnection and tell it to use the IOM provider.  
OleDbConnection conn = new OleDbConnection();  
conn.ConnectionString = "Provider=sas.IOMProvider; SAS Workspace ID=" +  
    sasWS.UniqueIdentifier ;
```

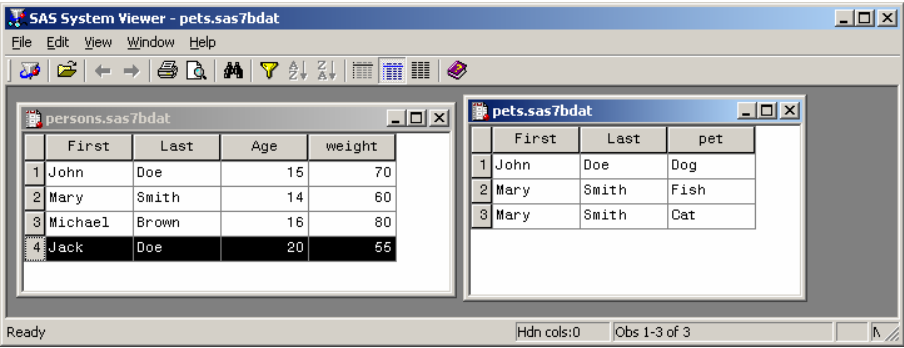
Step 4 – Setting SAS library path

Before working with data in the SAS datasets, set the SAS library path (libname) with the IOM provider. This can be done with an OleDbCommand as shown in the following code snippet. In our sample project, the SAS datasets reside in the directory C:\Temp\IOM. We also aliased it as MyLib.

```
// Set SAS dataset library path (libname)  
OleDbCommand cmd = new OleDbCommand() ;  
cmd.Connection = conn ;  
cmd.CommandType = CommandType.Text ;  
cmd.CommandText = @"libname MyLib 'c:\temp\IOM' " ;  
conn.Open() ;  
cmd.ExecuteNonQuery() ;  
conn.Close() ;
```

Step 5 – Retrieve SAS dataset into a DataSet

With ADO.NET, a SAS dataset can be treated like a relational table. It can be queried or updated with the standard SQL SELECT, INSERT, UPDATE and DELETE statements. Our test SAS datasets are shown below in Figure 4:



The screenshot shows the SAS System Viewer interface with two datasets displayed in table format. The left dataset, 'persons.sas7bdat', has columns First, Last, Age, and weight. The right dataset, 'pets.sas7bdat', has columns First, Last, and pet. The status bar at the bottom indicates 'Ready', 'Hdn cols:0', and 'Obs 1-3 of 3'.

	First	Last	Age	weight
1	John	Doe	15	70
2	Mary	Smith	14	60
3	Michael	Brown	16	80
4	Jack	Doe	20	55

	First	Last	pet
1	John	Doe	Dog
2	Mary	Smith	Fish
3	Mary	Smith	Cat

Figure 4 - Test SAS Datasets

The following code snippet retrieves observations (records) whose age is greater than 15 from the Persons dataset. Note that we must prefix the SAS dataset with the libname alias defined in step 4 in the SQL statement.

```
// Retrieve SAS data into a DataSet  
string sql = "select * from MyLib.Persons where age > 15" ;  
OleDbDataAdapter da = new OleDbDataAdapter(sql, conn) ;  
System.Data.DataSet ds = new System.Data.DataSet("SAS") ;
```

```
da.Fill(ds) ;
WriteDataSet(ds.Tables[0]) ;
```

The output of the above code snippet is shown below.

Michael	Brown	16	80
Jack	Doe	20	55

SQL statement with join can also be used to retrieve data from multiple SAS datasets as shown in the following code snippet.

```
// SQL statement with join
sql = "select Persons.First, Persons.Last, Persons.Age, Pets.Pet " +
      "from MyLib.Persons, MyLib.Pets " +
      "where Persons.First = Pets.First " +
      " and Persons.Last = Pets.Last " +
      "Order By Age " ;
da = new OleDbDataAdapter(sql, conn) ;
ds = new System.Data.DataSet("SAS") ;
da.Fill(ds) ;
WriteDataSet(ds.Tables[0]) ;
```

The output of the above code snippet is shown below.

Mary	Smith	14	Cat
Mary	Smith	14	Fish
John	Doe	15	Dog

We can also update the data in a SAS dataset with the SQL UPDATE statement:

```
// Update data in a SAS dataset with a SQL UPDATE
sql = "update MyLib.Persons set Age = 17 " +
      "where First = 'John' and Last = 'Doe' " ;
cmd = new OleDbCommand(sql, conn);
cmd.CommandType = CommandType.Text ;
conn.Open() ;
cmd.ExecuteNonQuery() ;
conn.Close() ;
```

SQL INSERT and DELETE are performed in a similar manner.

RUNNING SAS SCRIPT DYNAMICALLY WITHIN A .NET PROGRAM

Up to this point, we have only used the data component of the IOM to manipulate data in SAS datasets. The IOM also provides the LanguageService component to allow client applications to submit SAS code to the IOM server for execution as well as retrieve log and output listing. In addition to submitting SAS code, the LanguageService allows us to execute SAS Stored Processes – a special format of a SAS program available on the server. Using SAS Stored Processes, we can have centralized SAS programs that can be invoked by multiple SAS clients. Here we demonstrate the simpler method, submitting SAS code to the IOM server for execution².

² See the SAS support website (<http://support.sas.com/>) for more information on advance SAS stored processes

AD11

The code snippet below illustrates how to obtain a LanguageService object, and then use it to submit a simple SAS data step and retrieve its log stream.

```
// Submit SAS script
string script = @"libname mylib 'c:\temp\iom' ; " +
    @"data mylib.myds; " +
    @"    set mylib.persons ; " +
    @"    if (age <= 15) ; " +
    @"run; " ;
sasWS.LanguageService.Submit(script) ;
string outLog = sasWS.LanguageService.FlushLog(1000) ;
StringBuilder sb = new StringBuilder() ;
while (outLog.Length > 0) {
    sb.Append(outLog) ;
    outLog = sasWS.LanguageService.FlushLog(1000) ;
}
Console.WriteLine(sb.ToString()) ;
```

The screen capture of the SAS log is shown below.

```
NOTE: Copyright (c) 2002-2003 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) 9.1 (TS1M3)
      Licensed to RESEARCH TRIANGLE INSTITUTE, Site 0047670011.
NOTE: This session is executing on the XP_PRO platform.
NOTE: SAS Initialization used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds
NOTE: Libref MYLIB was successfully assigned as follows:
      Engine:           V9
      Physical Name: c:\temp\iom
1          libname mylib 'c:\temp\iom' ;
1          !                               data mylib.myds; set mylib.persons ;
if
(age <= 15) ;
1          ! run;
NOTE: There were 4 observations read from the data set MYLIB.PERSONS.
NOTE: The data set MYLIB.MYDS has 2 observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time          0.17 seconds
      cpu time           0.00 seconds
```

AD11

INTEGRATION OF SAS AND .NET FOR SIMPLE REGRESSION ANALYSIS

Integrating SAS with .NET also allows developers to greatly enhance some tedious processes. As an example, we have created a .NET program to execute multiple PROC REG procedures and then the program is used to further analyze the output of these procedures. Our example uses what is called a backward selection stepwise regression to find a model of best fit where the most parameters are significant.

Backward selection stepwise regression simply takes all the linear and cross product variables together and performs a regression procedure. The process requires systematically taking one variable away at a time and continuing to perform a regression at each iteration. Each iteration or variable removed will result in different regression results that could possibly be the best model fit. With .NET integration we were able to automate this time-consuming process by having .NET script our SAS code and analyze results without interaction from the user.

To perform the stepwise regression, a comma delimited CSV file having column headers as variable names and numerical values for each observation plus a Boolean option for obtaining natural logarithmic values are added as arguments to the IOM_DEMO.exe program. The syntax to start the regression process and an example of the Data.csv file follows.

```
C:\>IOM_DEMO.exe log=true file="c:\mydata\data.csv"
```

```
Consumption,Population,NumCars,Tax,Price,Income
```

```
2453941,4467461,4235089,18,133.4,35160
```

```
283509,632249,597734,8,161.2,57363
```

```
2493038,5295929,3967117,18,146.8,42704
```

```
1420446,2691581,1863798,21.7,132.4,33339
```

```
14966705,34526660,28780056,18.4,152.6,47262
```

```
2120606,4427016,4648820,22,142.9,49397
```

The first column of the file is the dependent variable and other columns are considered potential significant independent variables. The .NET code opens the file and extracts the header information for variable names and then starts scripting SAS code to import data, create variable names and perform log functions on variables if needed.

At the same time that all the variables are being scripted .NET is also keeping track of all the imported and newly created variables so that it can use them later to perform the regression analysis. Even though the SAS code is being created in .NET it is still only resident in memory and has yet to be passed to SAS for processing. Only after all the SAS syntax has been created will the text be submitted to SAS for processing using the IOM. A partial example of the SAS code created by .NET is shown below along with the .NET code that creates the custom SAS script:

```
data MyData;  
InFile "c:\mydata\Data.csv"  
Delimiter=', '  
FirstObs=2;  
input Consumption Population NumCars Tax Price Income run;  
lConsumption=log(Consumption);lPopulation=log(Population);lNumCars=log(NumCars  
);  
lTax=log(Tax);lPrice=log(Price);lIncome=log(Income);  
lIncome_lPrice=lIncome*lPrice; lIncome_lTax=lIncome*lTax;  
lIncome_lNumCars=lIncome*lNumCars; lIncome_lPopulation=lIncome*lPopulation;
```

.NET Code

```
private string createVariables(string x,bool y){
```

```

string modelVars=null;    //Our Model Variables Syntax
string createVars=null;   //Create Variables syntax
char[] splitter={' ',' '}; //Define our splitter variable
//check to perform Natural Log
if(y==true) {
    string []allVariablesArray=null;    //establish variable names array
    allVariablesArray=x.Split(splitter);
    //Create SAS syntax
    for(int z=0;z<=allVariablesArray.Length-1;z++){
        createVars = createVars + "1" + allVariablesArray[z] +
        "=log(" + allVariablesArray[z] + ")";
        //put a comma after each variable unless it is the last one
        if(z!=allVariablesArray.Length-1){
            modelVars=modelVars + "1" + allVariablesArray[z] + ",";
        }
        else{
            modelVars=modelVars + "1" + allVariablesArray[z] + "";
        }
    }
}
else {modelVars=x;}
string []modelVarsArray=null;
modelVarsArray=modelVars.Split(splitter);
//now that we have the array of modelvars we loop through them
for(int z=modelVarsArray.Length-1;z>=0;z--){
    //Create the syntax for creating the paired variables
    for(int r=z-1;r>=0;r--) {
        createVars=createVars + " " + modelVarsArray[z].ToString() +
        "_" + modelVarsArray[r].ToString() + "=" +
        modelVarsArray[z].ToString() + "*" +
        modelVarsArray[r].ToString()+"";
        //Assign the variables into modelvars to be used in regression
        modelVars=modelVars + "," + modelVarsArray[z].ToString() + "_" +
        modelVarsArray[r].ToString() + "";
    }
}
} //return the createVars syntax and the modelVars syntax
return createVars+"|"+modelVars;

```

}

Now that .NET has created the SAS script creating all the variables and the program also knows the name of all the variables required, the next logical step is to start scripting the analysis part of the integration. As was stated earlier, the backward stepwise regression technique performs a regression on all possible variables and then systematically removes independent variables one by one until there are no other variables left. A simple loop inside the program is used to work through all the variables to dynamically create the PROC REG procedure as more and more variables are removed.

Through each iteration of the loop the PROC REG procedure and the other memory-resident SAS syntax is passed to the IOM for execution in the SAS engine as shown below;

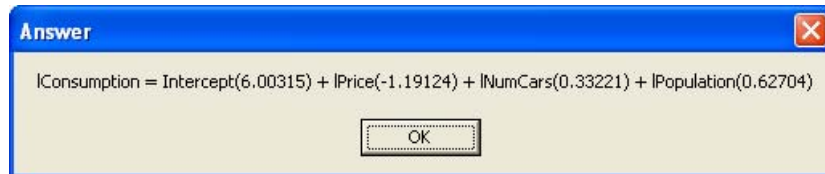
```
private void doSAS(string strInputVal, string strModelVal){
    //Establish the SAS Workspace
    SAS.Workspace sasWS = iom.SasWorkspace;
    //Create our SAS syntax header for importing
    //concatenate header,input,model and footer syntax
    string strHeader="data MyData;InFile \" + myGlobal.inputfile +
        "\" Delimiter=', ' FirstObs=2;";
    string strFooter="@run;ods XML close;";
    //SAS header and footer are within the scope of doSAS strModelVal including
    //syntax creating variables for the regression analysis
    StringBuilder sasScript = new StringBuilder() ;
    sasScript.Append(strHeader);
    sasScript.Append(strInputVal);
    sasScript.Append(strModelVal);
    sasScript.Append(strFooter);
    //submit the sasScript to the SAS engine for processing
    sasWS.LanguageService.Submit(sasScript.ToString()) ;
    //Capture information for log information
    string outLog = sasWS.LanguageService.FlushLog(1000) ;
    StringBuilder sb = new StringBuilder() ;
    //Loop through the log information and assign it to the string
    while (outLog.Length > 0){
        sb.Append(outLog);
        outLog = sasWS.LanguageService.FlushLog(1000);
    }
    //Output data to the form area for viewing
    txtLog.Lines = sb.ToString().Split('\n');
}
```

the resulting output is directed to an XML document on the client machine. This XML document, produced by the SAS Output Delivery System (ODS), contains all the needed statistical information to perform a comparison between regression models. In order to perform our comparison between these models the resulting XML document is parsed and parameter estimates, t values, and R square values are stored into a .NET storage framework called a datatable. These datatables are memory-resident and similar to a table in a database that

AD11

can be queried or updated in similar ways. Each regression model that is run has its resulting XML output stored into this datatable. Once all variables have been exhausted in the regression loop, our final step is to obtain the best fit model stored in the datatable.

The final step involves returning what is the best fit model stored in the datatable according to parameters hardcoded into the program. In this case we are assuming a 95% confidence interval for as many variables that will maximize the Adjusted R^2 value. This is done by executing a filter on the datatable itself. The filter, similar to a SQL query, reads where " $Pr>|t| < .05$ and $Adj\ R-Sq \neq 1$ order results by the iteration id". The resulting records from this query are further analyzed and where the iteration id has the most number of significant parameters we flag this as our best fit. Our resulting best fit parameter estimates are displayed in a pop-up as indicated here.



CONCLUSION

With .NET and SAS working together the statistical power of SAS can now be accessed by all software developers, not just the SAS experts. The integration opens the door for creating reusable SAS objects that can be incorporated into large-scale software development projects where SAS may have never been able to interact.

This paper has discussed how the interaction between the two technologies works and gives a detailed example of how integration could be used for many other types of procedures in the SAS system.

As many SAS users are aware, SAS Internet provides functionality through a browser, but the IOM described here takes this a step further and allows more powerful programming languages such as Java, C# and Visual Basic to harness the capabilities of the SAS engine. With more users able to utilize SAS and its tools the SAS community will be able to expand its reach to even more developers with a much larger array of talents. This can only complement the SAS suite of products and the community that it supports.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact any of the authors at:

Mai Nguyen
RTI International
800 Park Drive Offices
Research Triangle Park, NC 27709
Phone: (919) 541-8757
Email: mnguyen@rti.org
Web: www.rti.org

Shane Trahan
RTI International
800 Park Drive Offices
Research Triangle Park, NC 27709
Phone: (919) 541-5848
Email: srt@rti.org
Web: www.rti.org

Patricia Nguyen
RTI International
4506 S. Miami Blvd.
Suite 100 & 150
Durham, NC 27703
Phone: (919) 485-5713
Email: pnguyen@rti.org
Web: www.rti.org

Jonathan Cirella
RTI International
800 Park Drive Offices
Research Triangle Park, NC 27709
Phone: (919) 541-6815
Email: cirella@rti.org
Web: www.rti.org

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.