

Paper 011-2008

SAS® Integration Technologies, UNIX and Visual Basic .Net Integration Procedure

Antoine Chevrette, Statistics Canada, Ottawa, On

ABSTRACT

For years, the Survey of Employment, Payrolls and Hours (SEPH) Section of Statistics Canada's Labour Statistics Division has been using UNIX servers to meet the needs of its surveys and its clients. Because of the complexity of processing the data, using SAS on a UNIX server was considered to be the best choice to speed up data manipulation. To date, SAS/AF® has been used to develop the editing and process management graphical user interfaces (GUIs).

Over the years, however, we have received increasingly complex requests for data analysis and editing GUIs. Today, the task of providing our clients with high-quality products that satisfy their requirements is even more challenging.

The UNIX version of SAS/AF offers a very limited set of tools for building complex GUIs. Consequently, other ways of meeting the Division's operational requirements have been explored. The PC version of SAS/AF was tested, and although it is more flexible than the UNIX version, it too has limited capabilities.

SAS IT (Integration Technologies) together with Microsoft Visual Basic .Net (VB.Net) was also tested. This appears to be the best solution since it is capable of developing complex GUIs in a reasonable amount of time. In addition, VB.Net is much more widely used than SAS/AF, and it is much easier to find reliable resources for building GUIs.

This article focuses on two separate problems. First, we will look at integration, using a concrete example from our research in both SAS IT – specifically, the Integrated Object Model (IOM) server and the SAS/CONNECT® server – and Microsoft Visual Basic .Net. Second, we will propose a solution combining Visual Basic .Net and SAS to solve the problem of navigating large files (files with millions of data elements) in real time.

INTRODUCTION

There are a number of articles on using VB.Net with SAS [1,2,3,4,5,6,7]. While they explore specific aspects of VB.Net and/or SAS, they do not combine all the theories into one. For example, the articles by Jahn [4] and Patter [7] discuss the use of SAS with VB.Net in general terms. They provide a good overview of the theory, but that theory is not detailed enough to apply to more complex examples. In addition, none of the articles covers the use of SAS/CONNECT, even though using SAS/CONNECT with IOM makes it possible to take advantage of some features of VB.Net and UNIX servers without having to buy SAS Server.

In this article, we will look at a new approach using SAS/CONNECT and provide a general picture of how to integrate those technologies. Lastly, we explore a solution that can be used to view large data files quickly.

PROPOSED SOLUTIONS

Statistics Canada's analysts always want greater freedom in editing their data. VB.Net can provide that freedom since it has the capability of combining a number of tools from the Microsoft suite. We provide an example of a VB.Net application that makes it possible to edit SAS data from a UNIX server in an Excel spreadsheet. That example shows the flexibility and convergence of the various applications in VB.Net.

We explore two methods of building the application, one using a CONNECT server and the other using the IOM server.

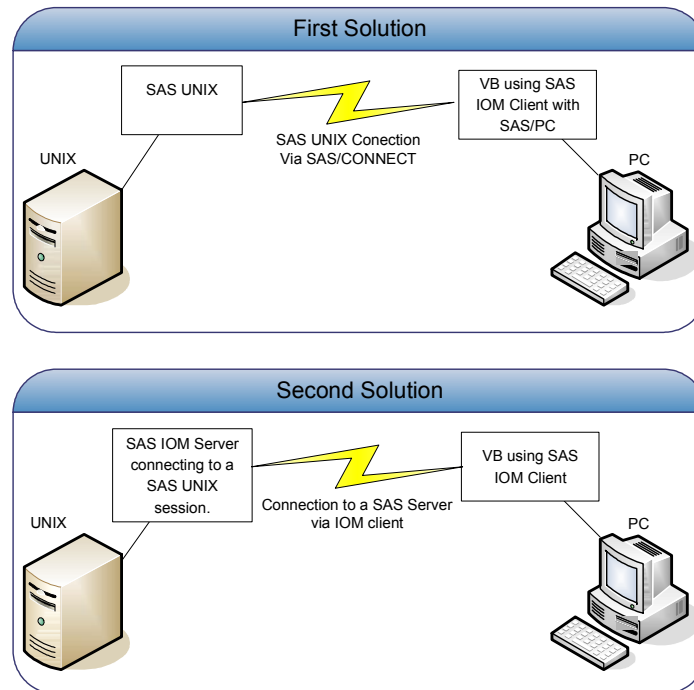
The first solution requires the use of the IOM client component, not the SAS IOM server. However, SAS/PC must be installed on the client machine to provide access to UNIX via SAS/CONNECT. This solution does not involve any additional costs if the organization already uses the basic SAS products for UNIX and PCs.

The second solution requires the use of SAS IT. First introduced in SAS version 8, SAS IT allows the integration of a client application with an SAS IOM server [1]. SAS IT supports industry standards (DCOM/CORBA). SAS IOM server with SAS IT must be purchased separately; it is not included in the basic version of SAS. In contrast, IOM client – the tool that mediates communications between the client and the SAS IOM server – is free and can be downloaded

directly from the SAS site. IOM client installs with an executable (inttech.exe). For this method, SAS/PC is not required on the client machine.

Visual Studio is needed to develop a VB.Net application. Visual Studio 2005 version 8 is used in this example. In creating our application, we need to add the references to the SAS objects (inttech.exe).

The code used to build the examples is provided in the body of the article. Some sections of code will be studied in detail, others will not. More information about the section of code that are not described in detail here can be found in the referenced articles.



GENERAL INFORMATION

The basic concepts are the same for both solutions. VB.Net should be regarded as an SAS user (a client), just like a human user. Since VB.Net does not use the keyboard or the SAS interface as a human user would, SAS provides a set of tools (IOM) that allow VB.Net to perform the same tasks. With those tools, VB.Net is able to open an SAS session, submit code, edit data, read reports, and so on.

SOLUTION 1

The first step before building the application is to create generic classes. To facilitate the transmission of information between SAS and VB.Net, a connection class is created. Since this solution uses SAS/CONNECT, a connection script must be provided for communication with the UNIX server. A manager class for that script must be created.

Since VB.Net is an SAS user, the connection class must have the following characteristics:

- 1- Opening of a SAS session on SAS/PC
- 2- Submission of SAS code on a PC and on UNIX
- 3- Editing of SAS data on UNIX

SasConnection

```
Imports System.IO

Public Class SasConnection
    Dim mySAS As SAS.Workspace
    Dim wm As New SASWorkspaceManager.WorkspaceManager
    Dim WithEvents ls As SAS.LanguageService
    Dim myConnection As New ADODB.Connection
```

```

Dim rs As New ADODB.Recordset
Dim serverName As String

Public Sub openWorkspace()
    Dim xmlInfo As String
    mySAS = wm.Workspaces.CreateWorkspaceByServer("",
        SASWorkspaceManager.Visibility.VisibilityProcess, Nothing,
        "", "", xmlInfo)
End Sub
Public Sub closeWorkspace()
    wm.Workspaces.RemoveWorkspaceByUUID(mySAS.UniqueIdentifier)
    mySAS.Close()
End Sub
Public Sub SubmitSASCode(ByVal strSASCode As String)
    Dim LS As SAS.LanguageService
    Dim arSource(1) As String
    LS = mySAS.LanguageService
    arSource(0) = strSASCode
    LS.SubmitLines(arSource)
End Sub
Public Sub RSubmitSASCode(ByVal strSASCode As String)
    Dim LS As SAS.LanguageService
    Dim arSource(1) As String
    LS = mySAS.LanguageService
    arSource(0) = "rsubmit;" & strSASCode & "endrsubmit;"
    LS.SubmitLines(arSource)
End Sub
Public Sub signon(ByVal libname As String)
    MsgBox(serverName)
    SubmitSASCode("options comamid=tcp remote=" & serverName & ";
        filename rlink 'connect.scr';signon;" & libname
        & "server=" & serverName & ";"")
End Sub
Public Sub singoff()
    SubmitSASCode("singoff;")
End Sub
Public Sub closeRS()
    rs.Close()
    myConnection.Close()
End Sub
Public Function openRs(ByVal table As String) As ADODB.Recordset
    myConnection.Open("provider=sas.iomprovider.1;sas workspace ID=" &
        mySAS.UniqueIdentifier, "", "", -1)
    rs.Open(table, myConnection, ADODB.CursorTypeEnum.adOpenDynamic,
        ADODB.LockTypeEnum.adLockOptimistic, &H200)
Return rs
End Function
Public Sub log()
    MsgBox(mySAS.LanguageService.FlushLog(10000000))
End Sub
Public Sub setservername(ByVal server As String)
    serverName = server
End Sub
End Class

```

In the connection class, the public function `OpenWorkspace` allows VB.Net to open an SAS session on SAS/PC. To do so, we have to create one workspace for each server in our SAS.Workplace (`mySAS`). (`CreateWorkspaceByServer` is a method of `SASWorkspaceManager`.) Since SAS Server is not being used, we have to write a server definition in the third argument of `CreateWorkspaceByServer` using the `Nothing` option. That option tells IOM to use SAS/PC as the server.

For a complete list of options for each method, see the articles by Silva [3], Key and Shamlin [5] and Eberhardt [6]. In contrast to solution 2, SASWorkspaceManager is used instead of SASObjectManager. SASWorkspaceManager does not allow us to connect to the metadata server [6], which is not used in this solution. SASWorkspaceManager is used for the sake of simplicity and versatility.

The next step is to create two public functions, one to submit SAS code on the PC (SubmitSASCode) and the other to submit SAS code on UNIX with SAS/CONNECT (RSubmitSASCode).

In addition, SAS data files are read and written using an ADODB connection by means of a Recordset [5]. The connection is made through SAS's iomprovider (see the public function openRS).

The log function displays the contents of the SAS execution log on the screen.

The signon function (SAS/CONNECT) uses a connection script built with the following class:

SasConnect Script

```
Imports System.IO
Public Class script
    Public script As String
    Public password As String
    Public username As String
    Private encryptPass As String

    Private Sub encrypt()
        Dim mySASconnection As New SasConnection
        mySASconnection.openWorkspace()
        mySASconnection.SubmitSASCode("filename file 'c:\windows\system32\
            pss.txt';" + _
            "proc pwencode in='" & password & "'
            out=file;" + _
            "run;")

        Dim temp As String() = File.ReadAllLines("c:\windows\system32\pss.txt")
        File.Delete("c:\windows\system32\pss.txt")
        encryptPass = temp(0)
        mySASconnection.closeWorkspace()
    End Sub
    Public Sub setUsername(ByVal user As String)
        username = user
    End Sub
    Public Sub setPassword(ByVal pass As String)
        password = pass
    End Sub
    Public Sub makeScript()

        Call encrypt()
        Dim writer As StreamWriter = File.CreateText("c:\windows\system32\
            connect.scr")
        'For the writer.writeline part see Appendix A
        writer.Close()
    End Sub
    Public Sub deleteScript()
        File.Delete("c:\windows\system32\connect.scr")
    End Sub
End Class
```

The encrypt function encrypts the user's password with SAS proc pwencode. The SubmitSASCode function of the connection class is used, which means that SAS/PC is responsible for encrypting the password. The password must be encrypted because it is written directly into the connection script file (see the makescript function).

At this point, the connection class and the makescript class are terminated. All the tools required to create the

application are now available.

The next step is to generate a new VB window containing an Excel spreadsheet (axSpreadSheet) and a button for submitting changes.

That is done by defining a few global variables in the window class.

```
Dim sasdata As String = "/user/home/"
Dim mySASconnection As New SasConnection
```

Everything that follows is executed when the window is loaded. First, we have to create the connection script.

```
Dim myscript As New script
myscript.setUsername(username)
myscript.setPassword(password)
myscript.makeScript()
```

Then we have to connect to UNIX.

```
mySASconnection.openWorkspace()
mySASconnection.setservername("lstdsas")
mySASconnection.signon("libname mywork" & sasdata)
```

By assigning a libname directly in signon, we create the libname permanently for the SAS session (see the use of mySAS.uniqueIdentifier in the connection class). However, we must redefine a libname when we call the RSubmit function, because the latter function is independent of the SAS/PC session. RSubmit calls SAS on UNIX directly, so all definitions set on the PC with IOM are unknown to SAS/UNIX.

The example below uses sashelp's company file and copies it to the user's home on UNIX.

```
mySASconnection.RSubmitSASCode
("libname mywork" & sasdata & ";" + _
" data mywork.company; " + _
"      set sashelp.company (where=(level2='NEW YORK')); " + _
" run;")
```

The spreadsheet is initialized as follows and assigned to the active worksheet owcSpread.

```
Dim owcSpread As AxOWC10.AxSpreadsheet
Dim owcWbook As OWC10.Workbook
Dim owcwsheet As OWC10.Worksheet

owcSpread = Me.AxSpreadsheet1
owcWbook = owcSpread.ActiveWorkbook
owcwsheet = owcWbook.ActiveSheet
```

In addition, the recordset that connects to an SAS dataset is defined as follows:

```
Dim rs As New ADODB.Recordset
rs = mySASconnection.openRs("mywork.company")
```

After the recordset is opened with the openRS function, it must be assigned to the Excel worksheet. Note that mywork was defined previously, when signon was called.

```
owcwsheet.Cells.CopyFromRecordset(rs)
```

A few parameters can be defined on the Excel worksheet to alter its appearance.

```
With AxSpreadsheet1
.AllowPropertyToolbox = False
.DisplayOfficeLogo = False
.DisplayTitleBar = False
```

```

.DisplayToolbar = True
With .ActiveWindow
    .DisplayColumnHeadings = False
    .DisplayRowHeadings = False
    .DisplayWorkbookTabs = False
    .DisplayHeadings = False
End With
End With

```

Following these steps, the connections must be closed.

```

mySASconnection.signoff()
mySASconnection.closeRS()
mySASconnection.closeWorkspace()

```

When the code above is executed, the data shown below will be displayed in the Excel worksheet. The Excel worksheet may have freeze panes, pivot tables, etc. In fact, any Excel feature can be used.

LEVEL2	LEVEL1	LEVEL5	DEPTH	LEVEL3	LEVEL4	JOB1	N
NEW YORK	International Aj	James Dargon	2	ADMIN	CONTRACTS	ASSISTANT	1
NEW YORK	International Aj	Natalie Besse	2	ADMIN	CONTRACTS	ASSISTANT	1
NEW YORK	International Aj	Emily P. Wallace	2	ADMIN	FINANCE	ASSISTANT	1
NEW YORK	International Aj	Deva K. Kumar	2	ADMIN	FINANCE	RESPONS. FINANC	1
NEW YORK	International Aj	Veronica Delorge	1	ADMIN	PERSONNEL	MANAGER	1
NEW YORK	International Aj	Danielle Prost	2	ADMIN	PERSONNEL	RECEPTIONIST	1
NEW YORK	International Aj	Elizabeth Cousteau	2	ADMIN	PERSONNEL	RESPONSIBLE	1
NEW YORK	International Aj	Herbert J. Kirk	2	ADMIN	PERSONNEL	RECEPTION	1
NEW YORK	International Aj	James H. Goodnight	1	ADMIN	SHIPPING	MANAGER	1
NEW YORK	International Aj	Barrett R. Joyner	2	ADMIN	SHIPPING	ASSISTANT	1
NEW YORK	International Aj	Sam Baker	1	SALES/MARKETING	MARKETING	MANAGER	1
NEW YORK	International Aj	Veronica Paulin	2	SALES/MARKETING	MARKETING	PROD. MAN. MATERN	1
NEW YORK	International Aj	Patricia Smith	2	SALES/MARKETING	MARKETING	ASSISTANT	1
NEW YORK	International Aj	Camille Besseel	2	SALES/MARKETING	SALES	LYON RESP.	1
NEW YORK	International Aj	Elaine Dumas	2	SALES/MARKETING	SALES	INDUSTRY	1
NEW YORK	International Aj	Alan Picard	2	SALES/MARKETING	SALES	RESPONS. TERTIA	1
NEW YORK	International Aj	Jean Francois Dumas	2	SALES/MARKETING	SALES	PUBLIC	1
NEW YORK	International Aj	Peter Caillon	2	SALES/MARKETING	SALES	AGENCE TERTIARE	1
NEW YORK	International Aj	Alan Bentz	2	SALES/MARKETING	SALES	ASSISTANT	1
NEW YORK	International Aj	Richard G. Roach	1	TECHN. SERVICES	MIS	MANAGER	1
NEW YORK	International Aj	Danielle Biabaut	2	TECHN. SERVICES	MIS	TECH.-CONS.	1
NEW YORK	International Aj	George H. Ruth	2	TECHN. SERVICES	MIS	TECH.-CONS	1
NEW YORK	International Aj	Michael Garris	2	TECHN. SERVICES	MIS	ASSISTANT	1
NEW YORK	International Aj	Claire Vixant	2	TECHN. SERVICES	MIS	TRANSI ATOUR	1

For example, if we have edited the LEVEL5 column, we have to press Enter and click on Submit Changes to capture the changes. Then the following code is executed:

```

mySASconnection.openWorkspace()
mySASconnection.setservername("lsdsas")
mySASconnection.signon("libname mywork" & sasdata)

Dim rs As New ADODB.Recordset
rs = mySASconnection.openRs("mywork.company")

```

The code below updates the SAS dataset on UNIX. This iteration could be constructed more optimally, by updating only the information that has changed.

```

Dim i As Integer
rs.MoveFirst()
i = 2
While Not rs.EOF
    rs("LEVEL5").Value =

```

```

        AxSpreadsheet1.ActiveSheet.Cells(i, 3).value()
        i = i + 1
        rs.Update()
        rs.MoveNext()
    End While

    MsgBox("update done")

    mySASconnection.singoff()
    mySASconnection.closeRS()
    mySASconnection.closeWorkspace()

```

A DataGridView could have been used as a substitute for the Excel worksheet.

```
Dim sasdataadapter As New System.Data.OleDb.OleDbDataAdapter
```

This object copies the contents of a recordset into a DataTable object.

```
Dim table As New DataTablesasdataadapter.Fill(table,
mySASconnection.openRs("mywork.company"))
```

Lastly, we have to assign the table to the DataGridView.

```
DataGridView1.DataSource = table
```

SOLUTION 2

Solution 2 uses the SAS IOM server, and SAS/PC is not required. Since a direct connection to UNIX is used (SAS Server runs directly on UNIX), queries run slightly faster than with solution 1.

For example, to open a session on the SAS/UNIX server, the following code can be used with the following connection class:

```
Dim mySASconnection As New SasConnection

mySASconnection.ServerName = "lsdsas"
mySASconnection.port = 6066
mySASconnection.login = login.username.Text
mySASconnection.password = login.password.Text
mySASconnection.open()

```

The connection to UNIX is much simpler. A connection script does not have to be created.

SasConnection

```
Imports System.IO

Public Class SasConnection

    Dim MyServer As String
    Private MyLogin As String
    Private MyPassword As String
    Dim ServerPort As Integer

    Dim rs As New ADODB.Recordset

    Public WithEvents sasLSevents As SAS.LanguageService
    Dim cnnIOM As New ADODB.Connection

    Dim obSAS As SAS.Workspace
    Dim obObjectFactory As New SASObjectManager.ObjectFactory
    Dim obServer As New SASObjectManager.ServerDef
    Dim objectkeeper As New SASObjectManager.ObjectKeeper()

```

```

Public Function open() As Boolean
    obServer.MachineDNSName = MyServer
    obServer.Protocol = SASObjectManager.Protocols.ProtocolBridge
    obServer.Port = ServerPort
    'Open as sas workspace with SASObjectManager.ObjectFactory
    On Error GoTo wrong

    obsAS = obObjectFactory.CreateObjectByServer("myServer", True, obServer,
                                                MyLogin, MyPassword)

    'Instantiate sasLSevents to be able to send sascode to the server
    sasLSevents = obsAS.LanguageService
    'Use the object keeper in order to assign a uniqueIdentifier to the adodb
    'connection.
    'This allows us to use every session definition (libname sasincl ...) in
    'the adodb connection
    objectkeeper.AddObject(1, "myServer", obsAS)
    'Open the connection via the sas iom provider
    cnnIOM.Open("Provider=sas.iomprovider.1;SAS workspace ID=" &
                obsAS.UniqueIdentifier, MyLogin, MyPassword)
    'Remove obsAS form the object keeper
    objectkeeper.RemoveObject(obsAS)
    Return True
wrong:
    Return False
    Exit Function
End Function
Public Sub closeWorkspace()
    obsAS.Close()
End Sub
Public Function openRs(ByVal table As String) As ADODB.Recordset
    'Open a recordset connection via cnnIOM
    rs.Open(table, cnnIOM, ADODB.CursorTypeEnum.adOpenDynamic,
            ADODB.LockTypeEnum.adLockOptimistic, &H200)
    Return rs
End Function
Public Sub closeRS()
    rs.Close()
End Sub
Public Sub SubmitSASCode(ByVal strSASCode As String)
    sasLSevents.Submit(strSASCode)
End Sub
Public Function log()
    Dim mytext As String
    mytext = sasLSevents.FlushLog(100000000)
    Return mytext
End Function
Property ServerName()
    Get
        ServerName = MyServer
    End Get
    Set(ByVal value)
        MyServer = value
    End Set
End Property
Property port()
    Get
        port = ServerPort
    End Get
    Set(ByVal value)
        ServerPort = value
    End Set

```



```

End Property
Property login()
    Get
        login = MyLogin
    End Get
    Set(ByVal value)
        MyLogin = value
    End Set
End Property
Property password()
    Get
        password = MyPassword
    End Get
    Set(ByVal value)
        MyPassword = value
    End Set
End Property
End Class

```

To provide SAS with the UNIX server information it needs, we have to use SASObjectManager.ServerDef (see the obServer variable in the connection class).

In this case, a metadata server¹ can be used since the class is created with SASObjectManager. The metadata server was not used in our research.

The procedures for submitting code and displaying and editing data are the same as in solution 1.

LARGE FILES

The use of large files may be problematic. Performance will be seriously degraded if a file exceeds the PC's memory capacity, since information is placed in active memory when it is read. However, this problem can be overcome using a virtual mode technique [8], which was tested and found to be effective.

The technique involves refreshing the data in a DataGrid each time the banner is accessed. This makes it possible to navigate SAS files with millions of data points in record time. The technique can be implemented with the three classes shown below (the code for the Dataretriever class which has been modified to connect to SAS is in Appendix B).

Just-in-time Data Loading.

The code was obtained from Microsoft. (<http://msdn2.microsoft.com/en-us/library/ms171624.aspx>)

Modifications have been made to the DataRetreiver class.

¹ The metadata server is a central warehouse for managing and storing metadata.

The Cache class determines which record the scroll bar is on and calls the DataRetriever class, which extracts the data from a dataset on UNIX. The SAS connection class from solution 2 is used.

The class may be used as indicated below.

First, we have to define memoryCache as an object of the Cache class.

```
Private memoryCache As Cache
```

The properties of the DataGridView are defined below.

```
With Me.DataGridView
    .Size = New Size(800, 250)
    .Dock = DockStyle.Fill
    .VirtualMode = True
    .ReadOnly = True
    .AllowUserToAddRows = False
    .AllowUserToOrderColumns = False
    .SelectionMode = DataGridViewSelectionMode.FullRowSelect
End With
```

We have to state the name of the table to be used (tablename) in DataRetriever and assign it an SAS connection. Then we have to tell the Cache class which DataRetriever it will use and how many records to display at one time (e.g., 100).

```
Dim retriever As New DataRetriever(tablename)
retriever.mySASconnection = mySASconnection
memoryCache = New Cache(retriever, 100)
```

Then we add the names of the columns for DataRetriever.

```
For Each column As DataColumn In retriever.Columns
    DataGridView.Columns.Add(column.ColumnName, column.ColumnName)
Next
```

Then we provide the number of data points.

```
Me.DataGridView.RowCount = retriever.RowCount
```

Now, each time the banner is accessed, the procedure below, which refreshes the data in the DataGridView with the memoryCache.RetrieveElements function, is called.

```
Private Sub dataGridView2_CellValueNeeded(ByVal sender As Object, ByVal e As
                                         DataGridViewCellValueEventArgs)
    Handles DataGridView2.CellValueNeeded
    e.Value = memoryCache.RetrieveElement(e.RowIndex, e.ColumnIndex)
End Sub
```

CONCLUSION

As noted earlier, there are various methods of combining the VB and SAS technologies. The final choice must be based on the organization's needs.

With solution 1, it costs less to develop an application. The downsides are more complex code, the loss of some functions associated with SAS Server (metadata) and the need for SAS/PC on the client computer. These disadvantages are not enough to warrant the purchase of an SAS Server licence if the application to be built is straightforward. If the application is at all complex, it would be better to buy SAS Server.

It is important to keep in mind that using Visual Basic will not solve all the problems associated with developing an application that uses SAS. Some challenges and constraints have nothing to do with SAS. For example, viewing large files is not directly tied to SAS.

Whether SAS Server is used or not, integration of the SAS and VB.Net technologies allows for more rapid development of complex applications. An organization that already uses both technologies will certainly benefit by combining them.

REFERENCES

[1] – Scott Vodicka. 2000. Enterprise Integration Technologies, What is it and what can it do for me. Cary, NC: SAS Institute Inc, SUGI 25 paper 141-25.

[2] – Matthew Campbell. 2003. Using Visual Basic With SAS® To Produce Statistical Reports. Educational Testing Service, Princeton, NJ, NESUG 2003 PS013.

[3] – Greg Silva. 2003. Using IOM and Visual Basic in SAS® Program Development. Biogen Inc, Cambridge, MA, SUGI 28 paper 32-28.

[4] – Daniel Jahn. Developing an Open Client in Visual Basic. Cary, NC: SAS Institute Inc, White Paper

[5] – Darren Key, David Shamlin. 2002. Using SAS® Data To Drive Microsoft Office. Cary, NC: SAS Institute Inc, SUGI 27 paper 123-27.

[6] – Peter Eberhardt. 2003. SAS® in the Office: IT Works. Fernwood Consulting Group Inc., Toronto, ON, SUGI 28 paper 157-28.

[7] – Frederick Pratter. 2005. Access to SAS® Data Using the Integrated Object Model (IOM) in version 9.1. Eastern Oregon University, La Grande, OR, PharmaSUG 2005 AD05.

[8] – Implementing Virtual Mode with Just-In-Time Data Loading in the Windows Forms DataGridView Control. MSDN. (<http://msdn2.microsoft.com/en-us/library/ms171624.aspx>)

ACKNOWLEDGMENTS

I want to thanks Réal Laflotte and Isabel Philibert who gave me the time to experiment and research on the subject. I want to thanks Yves DeGuire who encourage me to write this paper. I also want to thanks Michael Wenzowski, Yves DeGuire, Peter Law and Karine Désilets for helping me review and edit this paper and to have fully supported me through the entire process. Thanks!

CONTACT INFORMATION

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

Antoine Chevette
Statistics Canada
R.H. Coats 100 Tunney's Pasture Driveway
Ottawa Ontario, K1A 0T6
613-951-9903
antoine.chevette@statcan.ca

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.

APPENDIX A

The .writeline portion of the SAS/CONNECT script

```

writer.WriteLine("    if not tcp then goto notcp;")
writer.WriteLine("    if signoff then goto signoff;")
writer.WriteLine("    waitfor    'login:')")
writer.WriteLine("                , 'Username:')")
writer.WriteLine("                , 'Scripted signon not allowed' : noscript")
writer.WriteLine("                , 120 seconds: noinit;")
writer.WriteLine("    type " + "" + username + ";")
writer.WriteLine("    type LF;")
writer.WriteLine("    type " + "" + encryptPass + ";")
writer.WriteLine("    type LF;")
writer.WriteLine("unx_log:")
writer.WriteLine("    waitfor 'Hello>'                : unxspawn")
writer.WriteLine("            , '$'")
writer.WriteLine("            , '>'")
writer.WriteLine("            , '%'")
writer.WriteLine("            , '}'")
writer.WriteLine("            , 'Login incorrect'    : nouser")
writer.WriteLine("            , 'Enter terminal type' : unx_term")
writer.WriteLine("            , 'TERM'                : unx_term")
writer.WriteLine("            , 30 seconds            : timeout")
writer.WriteLine("            ;")
writer.WriteLine("    log 'NOTE: Logged onto UNIX... Starting remote SAS now.');"
writer.WriteLine("    Type '/software/sas913/sasexe/sas -dmr -comamid tcp -device grlink")
noterminal -nosyntaxcheck' LF;")
writer.WriteLine("    waitfor 'SESSION ESTABLISHED', 90 seconds : nosas;")
writer.WriteLine("    log 'NOTE: SAS/CONNECT conversation established.');"
writer.WriteLine("    stop;")
writer.WriteLine("unxspawn:")
writer.WriteLine("    Type '/software/sas913/sasexe/sas -dmr -comamid tcp -device grlink")
noterminal ';"
writer.WriteLine("    Type '-nosyntaxcheck' LF;")
writer.WriteLine("    waitfor 'SESSION ESTABLISHED', 90 seconds : nosas;")
writer.WriteLine("    stop;")
writer.WriteLine("signoff:")
writer.WriteLine("    waitfor '$'")
writer.WriteLine("            , '>'")
writer.WriteLine("            , '%'")
writer.WriteLine("            , '}'")
writer.WriteLine("            , 30 seconds")
writer.WriteLine("            ;")
writer.WriteLine("    Type    'logout' LF;")
writer.WriteLine("    log 'NOTE: SAS/CONNECT conversation terminated.');"
writer.WriteLine("    stop;")
writer.WriteLine("unx_term:")
writer.WriteLine("    Type 'tty' LF;")
writer.WriteLine("    goto unx_log;")
writer.WriteLine("timeout:")
writer.WriteLine("    log 'ERROR: Timeout waiting for remote session response.');"
writer.WriteLine("    abort;")
writer.WriteLine("nouser:")
writer.WriteLine("    log 'ERROR: Unrecognized userid or password.');"
writer.WriteLine("    abort;")
writer.WriteLine("notcp:")
writer.WriteLine("    log 'ERROR: Incorrect communications access method.');"
writer.WriteLine("    log 'NOTE: You must set OPTIONS COMAMID=TCP; before using this;")
writer.WriteLine("    log '    script file.');"
writer.WriteLine("    abort;")
writer.WriteLine("noinit:")
writer.WriteLine("    log 'ERROR: Did not understand remote session banner.');"
writer.WriteLine("    nosas:")
writer.WriteLine("    log 'ERROR: Did not get SAS software startup messages.');"
writer.WriteLine("    abort;")
writer.WriteLine("noscript:")
writer.WriteLine("    log 'ERROR: Scripted signons are not allowed.');"
writer.WriteLine("    log 'NOTE: Clear any script file reference and retry SIGNON.');"
writer.WriteLine("    abort;")

```

DataRetriever Class

```

Public Class DataRetriever

    Implements IDataPageRetriever

    Private tableName As String

    Public mySASconnection As New SasConnection
    Private myRS As New ADODB.Recordset
    Dim table As New DataTable
    Dim adapter As New OleDb.OleDbDataAdapter

    Public Sub New(ByVal tableName As String)
        Me.tableName = tableName
    End Sub

    Private rowCountValue As Integer = -1

    Public ReadOnly Property RowCount() As Integer
        Get
            ' Return the existing value if it has already been determined.
            If Not rowCountValue = -1 Then
                Return rowCountValue
            End If
            Dim rs As ADODB.Recordset
            mySASconnection.SubmitSASCode("proc contents data=" + Me.tableName + "
                                         out=mynbobs; run;")
            rs = mySASconnection.openRs("mynbobs")

            rowCountValue = CInt(rs("nobs").Value)

            mySASconnection.closeRS()
            Return rowCountValue
        End Get
    End Property
    Private columnsValue As DataColumnCollection

    Public ReadOnly Property Columns() As DataColumnCollection
        Get
            ' Return the existing value if it has already been determined.
            If columnsValue IsNot Nothing Then
                Return columnsValue
            End If
            ' Retrieve the column information from the database.
            mySASconnection.SubmitSASCode("data header;" + _
                                         " set " + Me.tableName + "(obs=0);" +
                                         "run;")

            Dim table As New DataTable()
            myRS = mySASconnection.openRs("header")
            adapter.Fill(table, myRS)

            columnsValue = table.Columns
            mySASconnection.closeRS()
            Return columnsValue
        End Get
    End Property

    Private commaSeparatedListOfColumnNamesValue As String = Nothing

```

```

Private ReadOnly Property CommaSeparatedListOfColumnNames() As String
    Get
        ' Return the existing value if it has already been determined.
        If commaSeparatedListOfColumnNamesValue IsNot Nothing Then
            Return commaSeparatedListOfColumnNamesValue
        End If

        ' Store a list of column names for use in the
        ' SupplyPageOfData method.
        Dim commaSeparatedColumnNames As New System.Text.StringBuilder()
        Dim firstColumn As Boolean = True
        For Each column As DataColumn In Columns
            If Not firstColumn Then
                commaSeparatedColumnNames.Append(", ")
            End If
            commaSeparatedColumnNames.Append(column.ColumnName)
            firstColumn = False
        Next

        commaSeparatedListOfColumnNamesValue = _
            commaSeparatedColumnNames.ToString()
        MsgBox(commaSeparatedListOfColumnNamesValue)
        Return commaSeparatedListOfColumnNamesValue
    End Get
End Property
' Declare variables to be reused by the SupplyPageOfData method.
Private columnToSortBy As String

Public Function SupplyPageOfData( _
    ByVal lowerPageBoundary As Integer, ByVal rowsPerPage As Integer) _
    As DataTable Implements IDataPageRetriever.SupplyPageOfData

    ' Store the name of the ID column. This column must contain unique
    ' values so the SQL below will work properly.
    ' Store the name of the ID column. This column must contain unique
    ' values so the SQL below will work properly.
    If columnToSortBy Is Nothing Then
        columnToSortBy = Me.Columns(0).ColumnName
    End If

    Dim higher As Integer
    higher = lowerPageBoundary + rowsPerPage

    Dim table As New DataTable

    mySASconnection.SubmitSASCode(" data temp;" + _
        "set " + Me.tableName + "( where=(" +
        lowerPageBoundary.ToString + " < n <= " + (higher).ToString + ");" + _
        "run;")

    myRS = mySASconnection.openRs("work.temp")
    adapter.Fill(table, myRS)
    mySASconnection.closeRS()

    Return table

End Function
End Class

```