Paper 040-2007

An approach for a scalable, platform independent, automated solution using SAS IOM

Srinivas Bhat, sanofi-aventis U.S., Bridgewater, NJ
Vijay Venugopal, sanofi-aventis U.S., Bridgewater, NJ
Jim Fang, sanofi-aventis U.S., Bridgewater, NJ

ABSTRACT

Harnessing the functionality of SAS/SAS Connect and utilizing the flexibility of SAS Integrated Object Model (IOM), data can easily be moved between multiple platform independent applications like Microsoft Excel and SAS.

This paper presents a practical step-by-step approach to sourcing data from an Excel sheet to create a SAS dataset on the UNIX server. It further elaborates the use of SAS Call Execute data step function to transport the SAS dataset row-by-row as parameter to a SAS macro culminating in creation and FTP of CSV files to different servers.

The paper also highlights merits of the above approach in the areas of maintenance, scalability, ease of use and process automation.

INTRODUCTION

Assimilating disparate sources of data from multiple platforms in to a SAS platform to do data analysis presents various challenges. Firstly, it requires the understanding of multiple programming languages in order to integrate data. Next, it is expensive and less scalable to maintain disparate processes due to multiple points of failure. Third, it is resource intensive and leaves less opportunity for process automation.

To overcome all the above shortcomings, we have researched and presented a flexible approach using SAS Integrated Object Model (SAS IOM), SAS Call execute procedure and the usage of shell scripting to transport data to a SAS platform.

THE RECIPE:

STEP 1: DYNAMIC CREATION OF SAS CONTROL DATASET USING SAS IOM AND EXCEL VBA

In data analysis, it is not uncommon to cater to requests having disparate input parameters like mapping files in SAS, control files in Excel format which feed into new or existing scenarios.

This can easily be achieved using SAS integrated Object Model (IOM), Microsoft Active Data Objects (ADO) and Excel Visual Basic for Applications (VBA) which converts the Excel sheet into a SAS dataset. Maintenance of this Excel sheet can reside with a non-SAS savvy resource.

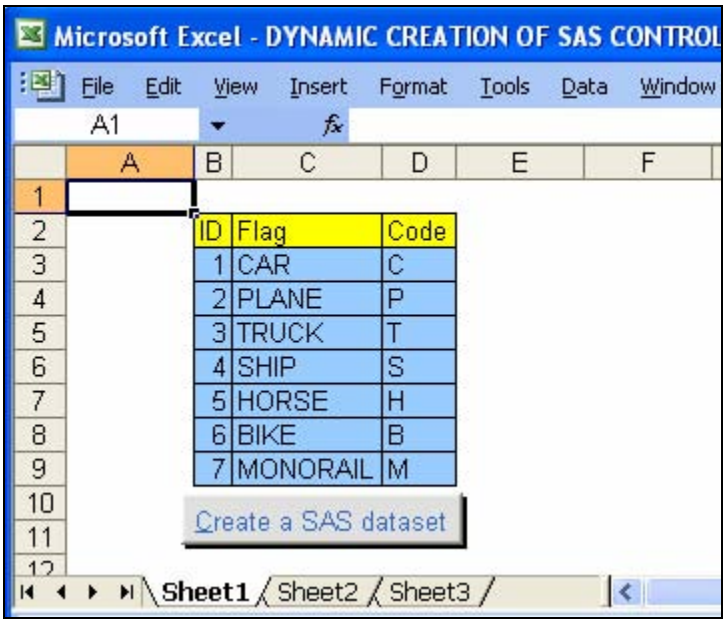

Figure 1

As depicted in Figure 1 above, there are three columns namely: ID, Flag and Code with some data and a button. Creating a SAS dataset from this sheet can be achieved by simply invoking the SAS Import procedure. But using technologies such as SAS IOM in conjunction with Microsoft ADO gives more control to the programmer to further process the scenario. Once parameterized, this technique is reusable by the business analyst without any programmatic changes. See appendix 1 for code snippet.

STEP 2: PROCESSING EACH ROW OF A SAS DATASET BY PASSING IT TO A MACRO (AS AN ORACLE CURSOR)

Programmers familiar with traditional Relational Database Systems (RDBMS) might feel the need of CURSOR like capability in SAS. This is available as a data step function called CALL EXECUTE in SAS.

From Step 1 above, we have created a SAS dataset in the UNIX server. Supposing one wants to filter for different types of vehicle models which are represented as rows in the defined dataset, we can set up a macro to calculate the fuel consumption and invoke this macro by reading the dataset and using the CALL EXECUTE data step function.

```
/* Macro to filter data and write to a .csv file */
%macro filter_veh(vehicle, code);
Data tmp;
   Set vehicles(where=(type=&vehicle));
Run;

/* Outputting to a .csv */
proc export data=tmp outfile="file&code"
dbms=csv replace;
run;
%mend;

/* Invoking the above defined macro for each record of the input dataset, passing
the columns as parameters to the macro. Note that the macro will be executed after
all the records have been read */

Data _null_;
Set libname.param(where=(code in (C P S R));
Call execute("%"||"filter_veh("||flag||","||code||");");
Run;
```

This process can be automated by setting up a CRON in UNIX to keep looking for a dataset which is created from STEP 1. As soon as the dataset is created the CRON executes a SAS program which reads the input dataset and passes each row of the dataset as a parameter to a macro. This macro filters all models belonging to the vehicle type and creates a .CSV file.

STEP 3: AUTOMATED FTP AND EMAIL NOTIFICATION OPTIONS

The CSV file created in the above step can be transferred automatically to different servers. Summary information can be communicated to multiple user groups via automated email confirming the culmination of the process. This can be achieved using native SAS functions or invoking UNIX shell commands from SAS as described in the below code.

```
/* Executing a shell script which will ftp the earlier created .csv files */
Data _null_;
Call system("ftp_put_files file&code>> ftplog");
Run;

/* Creating a file called mailbody using the UNIX commands and using the UNIX
mailx feature sending the email to desired users */

     data _null_;
     call system("echo All, > mailbody");
     call system("echo The following files have been ftped to the New1 server:
, >> mailbody");
     call system("echo  >> mailbody");
     call system("cat ftplog >> mailbody");
     call system("echo  >> mailbody");
```

```
        call system("echo SAS Team. >> mailbody");

        command = "mailx -s ""Files FTPed to New1 server"" fname.lname@mycompany.com
    second.name@mycompany.com < mailbody";
        call system(command);
        call system("rm mailbody ftplog");
    run;
```

The shell script to FTP files is shown below:

```
#! /bin/sh
   filenam=$1;
    echo $filenam
    # setting up the host, user and passwd
    HOST='199.199.99.01'
    USER='guest'
    PASSWD='guest'

    # logging on to the host for FTP by providing the user id and passwd

    ftp -nv $HOST <<EOF
    quote USER $USER
    quote PASS $PASSWD
    ascii
    prompt

    lcd /raw/out
    put $1
   quit
   EOF
```

CONCLUSION

Transporting disparate sources of data from multiple platforms in to a SAS platform to do data analysis requires the understanding of multiple programming languages. This paper describes a flexible, scalable and an automated approach to transform data into SAS. The approach is very simple and can be extrapolated to suit multiple scenarios.

REFERENCES

SAS Institute 2003. "SAS OnlineDoc"
http://support.sas.com/91doc/docMainpage.jsp

SAS Institute 2003. SAS Integration Technologies: Version 9 Documentation.
http://support.sas.com/rnd/itech/library/library9.html.

SAS Institute 2003. SAS Integration Technologies: Release 8.2 Documentation.
http://support.sas.com/rnd/itech/library/library82.html

Ted Conway. "Make Bill Gates and Dr. Goodnight Run Your SAS Code" *Proceedings of the Thirty first Annual SAS® Users Group International Conference. April 2004*
Using VBA, ADO and IOM to Make Excel and SAS Play Nice - http://www2.sas.com/proceedings/sugi30/157-30.pdf

ACKNOWLEDGMENTS

The authors acknowledge the leadership team at sanofi-aventis for their encouragement while the investigation was underway and support in completing the paper.

CONTACT INFORMATION

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

Name:	Srinivas Bhat	Vijay Venugopal	Jim Fang
Enterprise:	sanofi aventis	sanofi aventis	sanofi-aventis
Address:	55 Corporate Drive	55 Corporate Drive	55 Corporate Drive
City, State ZIP:	Bridgewater, NJ 08807	Bridgewater, NJ 08807	Bridgewater, NJ 08807
Work Phone:	(908)981-6087	(908)981-6115	(908)981-6080
E-mail:	Srinivas.Bhat@sanofi-aventis.com	Vijay.Venugopal@sanofi-aventis.com	Jim.Fang@sanofi-aventis.com

All Results were obtained on the multi-user and multi-CPU HP UNIX server running SAS 9.1.3. Actual numbers may vary depends on the server load.

APPENDIX

EXCEL VBA SAMPLE CODE

```
    ' This Object represents a session with the SAS system.
    '
    ' Note that this statement only declares a variable to reference a workspace.
    ' It does not create a workspace or assign a workspace to the variable.
    ' For creating workspaces, the SAS Integration Technologies Client provides
    ' the Workspace Manager. The Workspace Manager is an ActiveX component which can
    ' be used to create SAS Workspaces, either locally or on any server that
    ' runs the SAS Integration Technologies product.

    Dim obWS As SAS.Workspace

    Dim obWSM As New SASWorkspaceManager.WorkspaceManager
    ' By using the keyword new, you instruct COM to create a SAS Workspace Manager
    ' and assign a reference to it when the object variable (obWSM in this case)
    ' is first used.

    Sub SAS_Update()
    ' This sub SAS_Update is assigned to the Ok button in the Excel sheet

    On Error GoTo update_error
    Dim errorString As String
    Dim LastRow As Integer
    Dim theRng As Range
    Dim ChkFoundAtLeastOneFlag As Boolean

    ChkFoundAtLeastOneFlag = False

    LastRow = Cells.Find(What:="*", _
          SearchDirection:=xlPrevious, _
          SearchOrder:=xlByRows).Row
    Set theRng = Range(Cells(1, 2), _
        Cells(LastRow, 2))

    Rem start the SAS session
    ' The Workspace Manager has a Workspaces collection which has a
```

```
' CreateWorkspaceByServer method for creating a new workspace.

Set obWS = obWSM.Workspaces.CreateWorkspaceByServer("Local", _
           VisibilityProcess, Nothing, "", "", errorString)

Dim TargetRange As Range

Application.ScreenUpdating = False

Sheets("sheet1").Select

''Calling a function Named DetemineUsedRange to get the no of rows and columns
from workbook
DetermineUsedRange TargetRange

Dim sqlstr As String
Dim ColLen As Integer
Dim ColVal As String
Dim ColType As Boolean

'' Building the SQL string which will on execution create a table mapping in the
library lib1

sqlstr = "proc sql; Create table lib1.mapping ("
For ColKount = 1 To TargetRange.Columns.Count
    '' invoking the getMaxLengthOfCell func which returns the largest length of
all rows in that column
    ColLen = getMaxLengthOfCell(TargetRange.Columns(ColKount), ColKount)
    ColVal = Replace(TargetRange.Cells(1, ColKount), " ", "_")
    ColType = IsNumeric(TargetRange.Cells(2, ColKount))

    If ColKount <> TargetRange.Columns.Count Then
        If Not ColType Then
            sqlstr = sqlstr & ColVal & " char(" & ColLen & "), "
        Else
            sqlstr = sqlstr & ColVal & " num, "
        End If
    Else
        If Not ColType Then
            sqlstr = sqlstr & ColVal & " char(" & ColLen & ") "
        Else
            sqlstr = sqlstr & ColVal & " num "
        End If
    End If

Next
sqlstr = sqlstr & "); quit;"

Rem submit some SAS code

'' By invoking the macro auto_log we are defining the Unix server name, it's IP
address, pointing the location of the rlink

'' and more.

obWS.LanguageService.Submit "%auto_log; run;"

'' defining the libname lib1 on the unix server prodserv and executing the proc
sql built in the SQLstr which is a string var
```

```
obWS.LanguageService.Submit "libname lib1 '/dir/sasdata/' server=prodserv; run;" _
       & sqlstr & " run; "

''MsgBox obWS.LanguageService.FlushLog(100000)
''MsgBox obWS.LanguageService.FlushList(100000)

'' Checking the log file for Error(s) and warning(s)

Dim LogLines() As String
Dim cc() As SAS.LanguageServiceCarriageControl
Dim lt() As SAS.LanguageServiceLineType
Dim EachLineFromLogFile As Variant
Dim ErrList As String
Dim ChkFoundErrWarn As Boolean

    ChkFoundErrWarn = False

''Do
    obWS.LanguageService.FlushLogLines 200, cc, lt, LogLines
    For Each EachLineFromLogFile In LogLines
        If InStr(1, EachLineFromLogFile, "ERROR") Or InStr(1, EachLineFromLogFile,
"WARNING") Then

            ErrList = ErrList & EachLineFromLogFile
            ChkFoundErrWarn = True
        End If
''        logfile.Append (s)
    Next
    If ChkFoundErrWarn Then
        MsgBox ErrList, , "Found Some Error(s) / Warning(s)"
    End If
''Loop Until LogLines < 20

Rem open an ADO connection to the data set

Dim obConn As New ADODB.Connection
Dim obRS As New ADODB.Recordset
Dim connString As String

connString = "provider=sas.iomprovider.1; SAS Workspace ID=" _
             + obWS.UniqueIdentifier

obConn.Open connString

obRS.Open "lib1.mapping", obConn, adOpenStatic, adModeReadWrite, _
adCmdTableDirect

For RowKount = 2 To TargetRange.Rows.Count
'    If TargetRange.Cells(RowKount, 1) Then

        obRS.AddNew
        For ColKount = 1 To TargetRange.Columns.Count
            obRS.Fields(ColKount - 1) = TargetRange.Cells(RowKount, ColKount)
        Next

 '   End If
Next
MsgBox "Data Uploaded.", vbInformation, "UPDATE COMPLETE"

Rem tidy-up
```

```
obRS.MoveNext
obRS.Close

update_error:
If Err.Number <> 0 Then
    MsgBox "NO Updates Done.  There was some error while updating Data.",
vbCritical, "ERROR"
End If

obConn.Close
obWS.Close
DoNothing:
'If Not ChkFoundAtLeastOneFlag Then
'    MsgBox "At least One flag needs to be selected before processing", , "SAS
Export Wizard"
'End If
Sheets("Sheet1").Select
Range("A1").Select
Application.ScreenUpdating = True
End Sub

Function getMaxLengthOfCell(ByRef rng As Range, ByVal intCol As Integer) As
Integer

    Dim intRow As Integer
    Dim intMaxLenght As Integer
    Dim strMaxLengthValue As String
    Dim intLength As Integer

    For intRow = 2 To rng.Rows.Count
            'Check if its last cell If yes then that get Max length
            If intRow = 2 Then
                intMaxLenght = Len(rng.Cells(intRow, 1))
                strMaxLengthValue = rng.Cells(intRow, 1).Value
            Else
                intLength = Len(rng.Cells(intRow, 1))
                If intLength > intMaxLenght Then
                    intMaxLenght = intLength
                    strMaxLengthValue = rng.Cells(intRow, 1).Value
                End If
            End If
     Next
     getMaxLengthOfCell = intMaxLenght
End Function
' This sub receives the empty argument "usedRng" and determines
' the populated cells of the active worksheet, which is stored
' in the variable "theRng", and passed back to the calling sub.
Sub DetermineUsedRange(ByRef theRng As Range)
Dim FirstRow As Integer, FirstCol As Integer, _
    LastRow As Integer, LastCol As Integer
On Error GoTo handleError
FirstRow = Cells.Find(What:="*", _
      SearchDirection:=xlNext, _
      SearchOrder:=xlByRows).Row
FirstCol = Cells.Find(What:="*", _
      SearchDirection:=xlNext, _
      SearchOrder:=xlByColumns).Column
LastRow = Cells.Find(What:="*", _
      SearchDirection:=xlPrevious, _
      SearchOrder:=xlByRows).Row
```

```
LastCol = Cells.Find(What:="*", _
      SearchDirection:=xlPrevious, _
      SearchOrder:=xlByColumns).Column
Set theRng = Range(Cells(FirstRow, FirstCol), _
    Cells(LastRow, LastCol))
handleError:
End Sub

%macro auto_log;
/* use either an IP address or node name  */

%let  prodserv=111.111.111.111; /* production server IP address */

options comamid=tcp remote=prodserv;
filename rlink 'C:\Program Files\SAS\SAS 9.1\connect\saslink\sad_unix.scr';
libname usasmac "C:\Documents and Settings\All Users\Documents\My SAS Files\9.1\";
options mstored sasmstore = usasmac;
signon;
libname uwork slibref=work server=prodserv;
run;
%mend;

/* End of Appendix  */
```