

# Various Ways to Link SAS Programs and Microsoft Excel

Peter K. Harrison, Monsanto Company, St. Louis, Missouri  
 Gary B. Wagner, Monsanto Company, St. Louis, Missouri

## Abstract

Many applications today are written in Microsoft Excel, then need to use the SAS® System to obtain accurate statistical reporting. This paper will discuss ways to pass data and control back and forth between Excel and SAS sessions. The use of Dynamic Data Exchange (DDE) will be compared to using SAS/ACCESS® Software for PC File Formats or the SQL Procedure Pass-Through facility. The techniques of using the SAS system as an OLE automation server or as a shelled process will be discussed. The problems of synchronizing separate processes will be addressed.

**Keywords:** OLE AUTOMATION DDE ODBC PCFF ACCESS MACRO VBA

## Introduction

The SAS System for Windows, Release 6.12 provides several ways to read and write data between the SAS System for Windows and Microsoft Excel. These include exchanging text files, Dynamic Data Exchange (DDE), OLE Automation, Open DataBase Connectivity (ODBC), and SAS/ACCESS® Software for PC File Formats. This paper will concentrate on using the SAS System as a background program to Excel. The methods discussed will provide Excel programmers with the means to obtain SAS results and display them within the context of an Excel application interface.

### How can a SAS program obtain values from Excel?

The simple way to exchange data between Excel and the SAS system is with flat files. This does not really require any changes to either program beyond normal input-output processes. An Excel sheet can be saved as text, either CSV or Tab delimited, and the SAS system can then read it.

```
[EXCEL]
ChDir "C:\Test"
ActiveWorkbook.SaveAs
    Filename:="C:\Test\data.csv",
    FileFormat:=xlCSV,
    CreateBackup:=False
```

```
[SAS]
data new;
file 'c:\test\data.csv' dlm=',' dsd
missover;
input first second third forth fifth;
run;
```

To read from comma-separated format (CSV), use a DLM=',' on the FILE statement and to read from a tab-delimited file, specify the option as DLM='09'X. By using the DSD option on the FILE statement, consecutive delimiters are interpreted as missing values. DSD also makes the delimiter default to a comma. In conjunction with the MISSEVER option, this allows you to bring in Excel data that contains some blank cells where data is missing.

This method will cover most of the simple needs where the Excel sheet can be saved and worked with as a text file, and where the SAS processing does not have to occur immediately or simultaneously.

However, there are times when the data exists in an Excel workbook, and the analysis needs are more immediate. In those instances, you may need to have the SAS system start processing the data as soon as it exists on the worksheet.

### Starting the SAS application from Excel

When timing is important, to control when the SAS job does its processing, the Excel application can Shell the SAS application.

```
[EXCEL]
Shell ("c:\sas\sas.exe -sysin
c:\test\test_call.sas")
```

Other options can be passed by the Shell command, such as directing the LOG or PRINT output of the SAS system

```
[EXCEL]
Shell ("c:\sas\sas.exe -config
c:\sas\config.sas -log c:\temp\saswork -
print c:\temp\saswork -sysin
c:\test\test.sas")
```

The following example sets up a SYSPARM string to pass into the SAS application

```
[EXCEL]
UserSAS = SASIntermediate & ".xls"
SASSysparm = "'Excel|[" & UserSAS & "]"SAS
Worksheet"

SASCommand = SASExeDrive & SASApplication
& " -sysin " & SASPrgrDrive &
SASProgramDir & SASProgram & " -sysparm "
& SASSysparm

SHELL(SASCommand)
```

Another method to integrate Excel and the SAS system is to shell the SAS application, then have the shelled SAS program initiate a Dynamic Data Exchange (DDE) request for the actual data as it sits in Excel. This DDE data can be obtained from the sheet or from a clipboard DDE triplet.

```
[SAS]
filename xlrc dde
'excel|[(dogdose.xls)skull!r2c1:r16c5];
run;

data skulls2;
infile xlrc;
input sex $ length basilar zygomatic
postorb ;

proc print;run;

/* read */

filename xlskuls dde
'excel|[(dogdose.xls)skull!partial];
run;
```

```
data skulls;
infile xlskuls;
input sex $ length basilar zygomatic
postorb ;

proc print;run;
```

The advantage to using Excel range names is that if data rows are inserted, a defined range will automatically extend in Excel. Using RC notation, the new number of rows or columns would need to be determined. To handle missing cells, we can use the DSD and MISOVER options to account for empty cells, specifying DLM='09'X and NOTAB. Using NOTAB tells the SAS system not to convert the tabs to blanks, then the DLM='09'x will use those tabs as delimiters

```
[SAS]
filename clipxl dde
'excel|[(book1.xls)sheet6!myarea4];
run;
data axl4;
infile clipxl dlm='09'x notab dsd
misover;
informat a $10. b $10. c $10. d $10.;
input a b c d;
run;
proc print data=axl4;
run;
```

DDE also provides a way to synchronize the actions of the Excel process, allowing you to alternate between Excel and SAS processing. The Excel program should Shell the SAS system, fall out of active processing at an End Sub, then do nothing until the SAS application has accomplished its tasks. When finished, the SAS application passes an Excel RUN request via DDE to start another Excel subroutine (perhaps called: 'SASReturn').

```
[SAS]
filename xlproc dde 'excel|system'; run;

data _null_; file xlproc;
put '[RUN("SASReturn")]';
```

## ODBC

If SAS/ACCESS is installed, then Open database connectivity (ODBC) allows read-only access to Excel spreadsheets (and read-write access to actual databases such as Microsoft Access or Dbase) via

**PROC SQL Pass-Through.** The SAS System for Windows release 6.12 works with System DSN data sources, which are local to a computer, not dedicated to a particular user. This capability is described in the SAS/ACCESS Software Changes and Enhancements: SQL Procedure Pass-Through Facility, Version 6 document.

An advantage to using ODBC is that Excel does not have to be running, thus freeing up system resources and memory. If an ODBC DSN has been established, then a SAS program can obtain data from a closed, inactive Excel sheet on disk. If multiple users are looking at the same Excel data in a LAN file share,

they will not all have to open Microsoft Excel to analyze the data, thus avoiding prompts as to whether they want to wait for read-write access or accept read-only access.

A disadvantage is that you must interactively use the 32-bit ODBC Manager to create or modify the ODBC data sources from the windows Control Panel.

### Setting up an ODBC connection

First, the Excel sheet must be defined to the ODBC manager (Fig. 1).

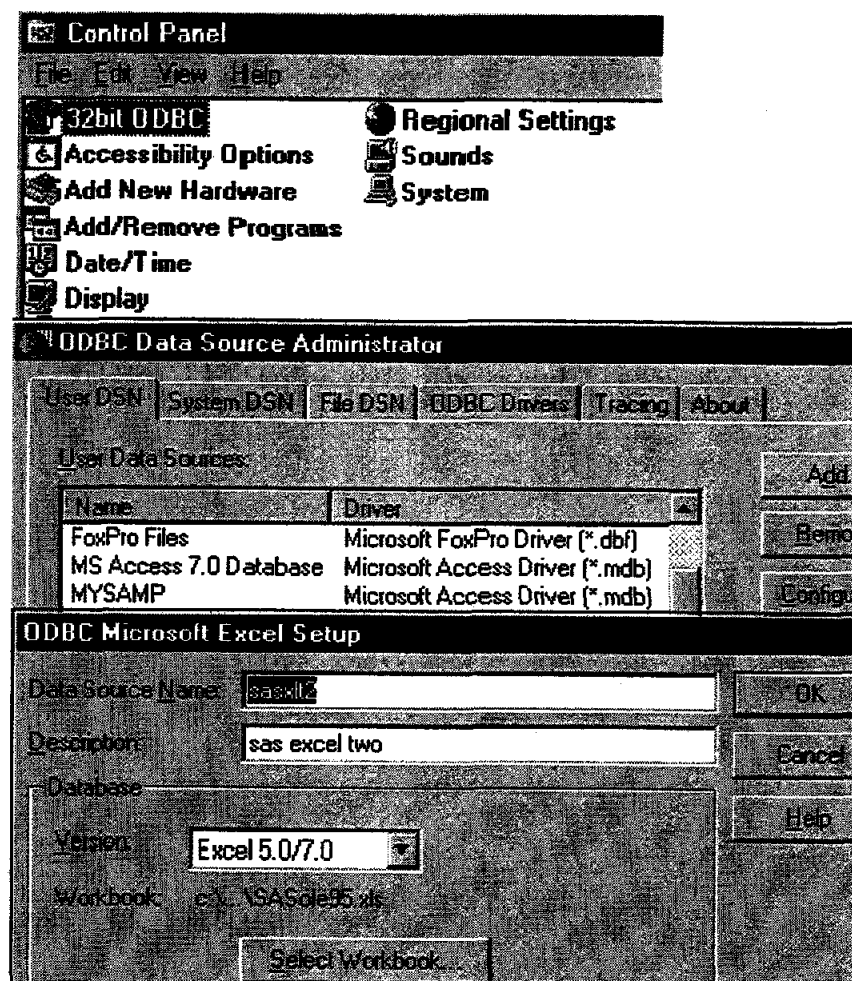


Figure 1. Setting up an ODBC DSN for an Excel workbook.

Once the ODBC DSN has been defined, you can use Proc SQL Pass-Through to read the Excel data from the workbook in the DSN. By having data ranges defined in the Excel workbook, the select statement can be directed to various data groups in Excel. In the following code, the query will select from the

"MYAREA4" named range in Excel. The default length for a character variable is 200.

```
[SAS]
proc sql; connect to odbc as xcel
(dsn=XLtest1);
```

```
select * from connection to xcel(select *
from myarea4 order by 2);
%put &sqlxmsg;
disconnect from xcel; quit;
```

## SAS/ACCESS Interface to PC File Formats

Similarly, a read-only access to Excel worksheets can also be accomplished using SAS/ACCESS Interface to PC File Formats (PCFF). This allows you to define a view describing the location of data in an Excel range. Each workbook should not contain more than 16 sheets, and the sheet names should be in sequence: Sheet1, Sheet2, Sheet3 etc... Both the access descriptor and the view descriptor have to be created in order to obtain Excel values. These are created in a SAS session but can remain across SAS sessions, if a permanent LIBNAME is used. This method allows variables to be mapped to new names and formats. Unlike the SQL procedure, however, there can be only one range accessed per descriptor; whereas multiple named ranges can be queried from the same ODBC DSN.

```
[SAS]
libname test 'c:\test\';

proc access dbms=xls;
create test.dog2.access;
path='c:\test\dogdose.xls';
range='dogtable';
worksheet='dogs';
getnames=y;
scantype=no;
skiprows=0;
assign=y;
list all;
```

```
create test.dog2.view;
select all; unique=y;
```

```
create test.skull2.access;
path='c:\test\dogdose.xls';
range='full';
worksheet='skull';
getnames=y;
scantype=no;
skiprows=0;
assign=y;
list all;
```

```
create test.skull2.view;
select all; unique=y;
```

```
create test.drug2.access;
path='c:\test\dogdose.xls';
range='drugdata';
worksheet='drugtest';
getnames=y;
scantype=no;
skiprows=0;
assign=y;
list all;
```

```
create test.drug2.view;
select all; unique=y;
```

```
run;
```

But SAS/ACCESS to PC File Formats also provides us with a way to write SAS data out to an Excel worksheet using PROC DBLOAD, which leads us into the next topic:

## How can Excel obtain values from SAS?

Excel can obtain results through flat files, DDE, ODBC, or a separate Excel worksheet created by PROC DBLOAD. It can also paste values from the Windows clipboard.

What does Excel do while the SAS code is executing? What does the SAS process do while Excel is processing? How can this be controlled and coordinated?

Excel can set up its data ranges or cell areas, shell the SAS system, then pause while the SAS program looks back to the Excel data ranges.

```
[SAS]
filename outall dde
&sysparm!sasresultsdata' notab; run;
```

```
data _null_; set here.all_res1;
cm='09'x; file outall;
```

```
put samp_no cm fit cm min cm max cm slope
cm ic50 cm rmse cm r_square;
run;
```

```
/* clear the dde definitions*/
filename outall clear;
```

In addition, by using DDE, a SAS application can request that Excel start a specific macro,

```
[SAS]
filename XLPROC DDE 'excel|system'; run;

data _null_;
file XLPROC;
put '[RUN("SASend")]';
run;
```

## PROC DBLOAD

The SAS system can create an Excel worksheet using the PROC DBLOAD method in PCFF.

```
[SAS]
libname test 'C:\TEST\';

proc dbload dbms=xls
data=test.dog2;
path='C:\test\dogsheet.xls';
accdesc=test.dogsht;
label;
putnames=y;
list all;
load;

run;

Proc dbload dbms=xls
data=test.skull2;
path='C:\test\skullsheet.xls';
accdesc=test.skullsht;
label;
putnames=y;
list all;
load;

run;

proc dbload dbms=xls
data=test.drug2;
path='c:\test\drugsheet.xls';
accdesc=test.drugsht;
label;
putnames=y;
list all;
load; run;
```

## OLE Automation

There is a way for Excel to cause individual actions to occur in SAS - OLE Automation. The SAS system for Windows is capable of being an OLE Automation Server. This allows SAS statements to be submitted

directly from Excel. Unfortunately, from the SAS system there is no explicit way to send result values for OLE clients, except through the clipboard. There is a roundabout way, however, by making use of the fact that DDE and OLE can be active at the same time. The requirement that Excel be active in order to process DDE requests is satisfied by the need to be active in order to submit SAS commands through the OLE interface. The passive read-only ODBC or PCFF methods of obtaining Excel values do not require that Excel be actually running.

Here is a mostly self-contained example to show OLE Automation and ODBC/DDE communication with Excel.

- 1) make sure that you have a c:\temp directory
  - 2) create ODBC dsn for 'sasxlt2' pointing to this workbook
  - 3) change values on Patt2 A1:B11 range
  - 4) change Sheet2!D1 value
- Tools-Macro-Run Calup

```
[EXCEL]
Sub calup()
Dim SASobj As Object 'Define the handle
to a SAS session
rc = DoEvents 'Allow time for
other things to finish

Set SASobj =
CreateObject("SAS.Application.612")
'Create the SAS session
flag_sas = True

SASobj.Visible = False 'Do not
want to see SAS
SASobj.Wait = True 'Wait
for SAS statements to complete

SASobj.submit ("options linesize=80;")
```

In the ODBC portion we populate a SAS data set (work.gibber) using ODBC from Excel to the SAS object. Note that the DSN sasxlt2 data source was predefined using ODBC manager to point to this workbook 'SASole95.xls' as a 'system' dsn. Patt2 is a sheet within the workbook, AA and BB named ranges on Patt2.

```
SASobj.submit (" proc sql;")
SASobj.submit (" connect to odbc as sxo
(dsn='sasxlt2');")
SASobj.submit ("create table gibber as ")
SASobj.submit (" select * from connection
to sxo (select * from ""Patt2$"" ")
SASobj.submit ("where AA>1 ); quit;")
```

The DDE input section reads the value in the named range (1 cell) PLATES on Sheet2 (!D1). We then define a filename, and use a \_null\_ data step to set a macro variable

```
SASobj.submit ("filename xlp1ts dde
'excel|sasole95.xls]sheet2!plates';
run;")

SASobj.submit ("data _null_;")
SASobj.submit ("infile xlp1ts;")
SASobj.submit ("input nplates $ ;")
SASobj.submit ("call symput('plates',
trim(nplates));")
SASobj.submit ("put 'nplates= ' nplates
;")
SASobj.submit ("run;")
```

This DDE output segment specifies the output rectangle with Excel R1C1 notation, turns off tab conversion and uses a \_null\_ data step to write the value of the macro variable 'plates' into sheet2!F1

```
SASobj.submit ("filename xlp1c dde
'excel|sasole95.xls]sheet2!r1c6' notab;
run;")

SASobj.submit ("data _null_;")
SASobj.submit ("file xlp1c;")
SASobj.submit ("pass=symget('plates');")
SASobj.submit ("put pass ;")
SASobj.submit ("run;")
```

The following code section is a DDE output example using a named range to specify the output rectangle with the Excel range name 'RESULTS', which is sheet2!A1:A11. Note that work.gibber does not contain 11 observations. You can save some time by submitting multiple statements at once.

```
SASobj.submit ("filename xlout dde
'excel|sasole95.xls]sheet2!results'
notab; run;")

SASobj.submit ("data _null_; set gibber;
file xlout; put bb ;run;")
```

Write a log file to disk

```
SASobj.command ("log;file
'c:\temp\sasole95.log' replace")
```

Now tell SAS to shutdown

```
SASobj.Quit
```

You must get rid of the handle to the SAS object, or it will continue as a ghost

```
Set SASobj = Nothing

End Sub
```

Return codes and messages can be returned directly to Excel via the setrc() function and captured with either object.RC or object.ResultString.

DDE and/or ODBC paths are still available to the SAS object under OLE control. The following two examples demonstrate these capabilities. The first example sends a command to the SAS system to include a separate SAS program file for execution. This file will contain a DDE command to start an Excel macro.

```
[EXCEL]
Dim sasobj As Object
```

```
Sub dogrun2()
MsgBox ("Dogrun2")
sas_final
End Sub
```

```
Sub dogrun1()
MsgBox ("Dogrun1")
sas_begin
sas_execs
End Sub
```

```
Sub sas_begin()
MsgBox ("sas_begin")
Set sasobj =
CreateObject("SAS.Application.612")
'Create the SAS session
sasobj.Visible = False
'Do not want to see SAS while running
sasobj.Wait = True
'Wait for each SAS statement to
complete
sasobj.submit ("options
linesize=80;")
End Sub
```

```

Sub sas_execs()
  MsgBox ("sas_execs")
  sasobj.submit ("filename xlskuls dde
'excel|[[dogdose.xls]skull!partial'; ")
  sasobj.submit ("run;")

  sasobj.submit ("data skulls;")
  sasobj.submit ("infile xlskuls;")
  sasobj.submit ("input sex $ length
basilar zygomat postorb ;")
  sasobj.submit ("run;")

  'include another SAS program
  sasobj.command ("pgm;include
'c:\test\prgram2.sas';submit")

End Sub

Sub sas_final()
  MsgBox ("sas_final")
  'Now tell SAS to shutdown
  sasobj.Quit
  'destroy the handle to the SAS object
  Set sasobj = Nothing
End Sub

```

The second example shows a previous code segment, with an ODBC connection to Excel established via the OLE automation interface. The resulting table is then partially written back to a second Excel worksheet.

```

[EXCEL]
SASobj.submit (" proc sql;connect to odbc
as sxo (dsn='sasxlt2');")
SASobj.submit ("create table gibber as
select * from connection to sxo (select *
from ""patt2$"" where aa>1 ); quit;")

SASobj.submit ("filename xlout dde
'excel|[[sasole95.xls]sheet2!results'
notab; run;")
SASobj.submit ("data _null_; set gibber;
file xlout; put bb ;run;")

```

### Synchronization Problems

A problem occurs with synchronization of process flow in the SAS System for Windows release 6.12 OLE automation. When an INCLUDE is submitted via OLE automation, the busy flag is reset before the included code is executed. This indicates to the calling program (Excel) that the SAS system is ready for processing more submit statements, when in fact it

is not. This can lead to Excel trying to print results that have not yet been output from the SAS application. Excel can however wait for the SAS system to send a DDE command to run the next procedure.

```

[SAS]
/* included example */
filename xlcmd dde 'excel|system';
run;

data _null_;
file xlcmd;
put '[RUN("Dogrun2")]';
run;

```

Attempts to use WIN32 API functions to handle synchronization using Windows DLL library calls such as CreateMutex or WaitForSingleProcess won't work well with OLE automation, because the SAS application is running as part of the Excel process. Having Excel wait for itself to be signaled results in a hung operating system. In this case we should back away from OLE automation, just to be able to have separate executable process spaces.

### Conclusions

Using SAS/ACCESS to obtain data from Microsoft Excel worksheets allows for common definitions of SAS data views and does not require that Excel be active. From the viewpoint of the Excel application developer; however, the ease of developing DDE read and write streams give the DDE method a much more powerful presence. OLE Automation provides a useful interface for small applications, but does not handle long running SAS processes well at this stage of its development.

### References:

Clegg, Jennifer and Rigsbee, Carol (1997), "OLE and the SAS System for Windows Release 6.12", *Proceedings of the Twenty-second Annual SAS Users Group International Conference*.

SAS Institute Inc. (1994), *SAS/ACCESS® Software Changes and Enhancements: SQL Procedure Pass-Through Facility Version 6*, Cary, NC: SAS Institute Inc. 44pp.

## *Tutorial and Solutions*

SAS Institute Inc. (1995),  
*SAS/ACCESS® Software for PC File  
Formats: Reference, Version 6*, Cary,  
NC: SAS Institute Inc. 226pp.

SAS Institute Inc. (1996), *SAS®  
Companion for the Microsoft Windows  
Environment*, Cary, NC: SAS Institute  
Inc., 302pp.

SAS and SAS/ACCESS are registered trademarks or  
trademarks of SAS Institute Inc. in the USA and other  
countries. The symbol ® indicates USA registration.

Other brand and product names are registered  
trademarks or trademarks of their respective  
companies.

The authors can be contacted as follows:

E-mail:

[peter.k.harrison@monsanto.com](mailto:peter.k.harrison@monsanto.com)

[gary.b.wagner@monsanto.com](mailto:gary.b.wagner@monsanto.com)

Mailing Address:

Monsanto Company  
700 Chesterfield Parkway North  
St. Louis, MO 63198