Using SAS as an Automation Server in Windows Application Development

Zhuan (John) Xu
Big Creek Software, LLC, Polk City, Iowa

ABSTRACT

Automation* is a mechanism through which one Windows application can control another application programmatically. This technology is currently widely used on the Windows 95/NT platform. Since Version 6.11, SAS can act as an automation server. Currently, SAS only exposes a few properties and methods. Even so, it enables us to develop applications that are fully functional. This paper describes how to develop Windows applications using SAS as an Automation server.

In addition, This paper presents Easy Interface for SAS, an application developed to simplify many routine SAS tasks, such as data set management, subsetting, sorting, importing/exporting text or Microsoft Excel files, common statistics, estimating sample size, random sampling, etc. Easy Interface for SAS is free and available for downloading at http://www.bigcreek.com/sas.

INTRODUCTION

SAS applications are developed with SAS/AF. Traditionally, SAS developers use Screen Control Language for application development. Newer applications are more likely built with the FRAME entry in SAS/AF, which takes advantage of object-oriented concepts, and newer interface techniques for Graphical User Interface (GUI).

Although a SAS/AF application has many advantages, such as tight integration with SAS/BASE and other SAS modules, there are times an alternative is desired. Most SAS users know how to code data steps, run many different SAS PROCs, and even do SAS MACRO programming. But not many know enough SAS/AF to turn a SAS program into to a GUI application. Since there is a steep learning curve with SAS/AF, it may not be practical for some users to learn it. On the other hand many organizations have developer experienced in Delphi, PowerBuilder, Visual Basic, Visual C++, or other rapid application development (RAD) tools. Utilizing these existing skills can be a very attractive option. As another example, you may want to enhance an existing non-SAS application by adding some analytic functionality only available in SAS. You can rebuild everything with SAS/AF or recreate the SAS functionality using the tool used to produce the original application. But it is certainly much attractive if we can integrate SAS into the existing application. In summary, it is highly desirable to use SAS as a database and analytic engine and integrate it with application developed using RAD tools.

There are different ways to develop such applications. For example, I have developed MAuditor, a decision support systems with SAS in the back end on a UNIX server and a PC database application developed with Borland Delphi at the front end. These two sides are connected together using the FTP protocol. MAuditor was presented at the 8th Annual Midwest SAS Users Group Conference. For more details, please see the MWSUG conference proceedings or contact the author for a copy.

In this paper, I will focus on another approach. That is, using SAS as an Automation server for Windows application development.

WHAT IS AUTOMATION?

Automation (formerly know as OLE Automation) is a mechanism through which one application (the automation client) can control another application (the Automation server). Automation is built on Microsoft's Component Object Model (COM) which also forms the basic for technologies such as OLE and ActiveX. Microsoft has also developed Distributed COM (DCOM) which allows Automation to work between applications running on two or more computer on a network. COM has become an integral part of the Windows platform.

Many Windows applications, including Microsoft Office applications, can act as Automation servers or clients. Development tools such as Microsoft Visual Basic and Borland Delphi make it fairly easy for Windows developers to create Automation servers and clients.

SAS AS AN AUTOMATION SERVER

Since version 6.11 SAS has been an Automation server. Through Automation, SAS exposes itself to other application as a programmable object. This means that we can use any application that can act as an Automation controller to create a SAS session and control it using the methods and properties that the SAS System makes available.

Currently, SAS only exposes a few properties and methods. They are listed below:

Method	Description
Command	Change which window receives the command
QueryWindow	Check whether specified window is open
Quit *	Quit SAS Automation process
Submit *	Submit to the SAS System for processing
Top	Bring the SAS session to the foreground

* Automation formerly know as OLE Automation. Microsoft renamed it recently.

Property	Description
Busy *	Indicates whether SAS is busy
CommandWindow	Sets the window that will receive any commands
CommandWindowVisible	Controls whether the Command Window is visible
	Sets the height of the SAS window
Parent	Sets the name of the parent window
RC *	Returns the return code passed by a user function.
ResultString *	Returns a string passed by a user function.
Title	Sets the AWS title
Visible *	Controls whether SAS is visible
Wait *	Controls whether Command execution is asynchronous or synchronous.
Width	Sets the width of the SAS window..
X	Sets the horizontal coordinate of the top left corner of the SAS
Y	Sets the vertical coordinate of the top left corner of the SAS

The SAS online help file contains more detailed description. Those marked with * are the methods and properties the author feels are most important and have been actually used. You can build an application without only these methods and properties. This certainly makes it very easy to program SAS as an Automation server. Even so, it enable us to develop applications that are fully functional.

To invoke a SAS session in Microsoft Visual Basic, we create a SAS Automation object:

```
Dim SAS as Object
Set SAS = CreateObject("SAS.Application")
```

It is very similar in Borland Delphi:

```
SAS :  Variant;
SAS := CreateOleObject('SAS.Application');
```

After SAS is started we can submit SAS programs. The following example in Visual Basic executes a simple DATA step:

```
SAS.Submit("data _null_; x=1; run;")
```

It is almost identical in Delphi:

```
SAS.Submit('data _null_; x=1; run;');
```

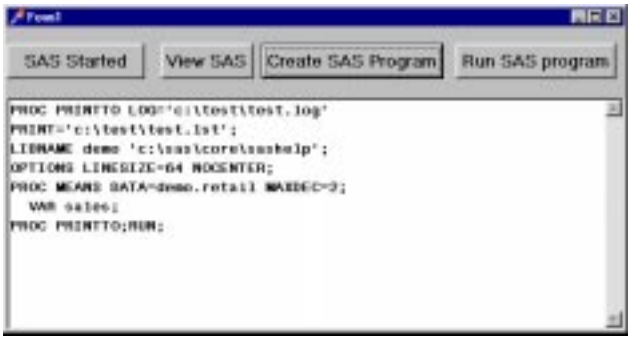
To run the SAS program `c:\test.sas`, we can replace the SAS code in quotes with `%include "c:\test.sas".` This is very much like running a SAS program in batch mode. The SAS output file and log file can be redirected with `proc printto`. The following code segment illustrate how to direct the output file into `c:\temp\test.lst` and the log file into `c:\temp\test.log`:

```
proc printto print='c:\temp\test.lst'
              log='c:\temp\test.log';
```

In turn, the output file can be used by a Delphi or a Visual Basic program to display the results. Similarly, the log file can be used for error checking and for extracting other useful information. Now, to create an application, all we have left to do is to generate the right SAS program, which is similar to the dynamic SQL commonly used in client/server development.

A SIMPLE APPLICATION

Now, let's take a look at a simple application. It was developed in Delphi for demo purposes. The following is the user interface:



You click on 'Start SAS' button to invoke a SAS session. After that, you can click on 'View SAS' to hide or show SAS windows. Clicking on 'Create SAS Program' will generate a SAS program and display it in the window. The SAS program has several steps. The first step is the PROC PRINTTO. It redirects the output file to `c:\test\test.lst` and the log file to `c:\test\test.log`. The second step creates a SAS library named 'demo' and points it to the directory `c:\sas\core\sashelp`. Note this is a directory created during the SAS installation and contains a SAS data set `retail.sd2`. The third step specifies some options. Note that specifying `LINESIZE` and `NOCENTER` is a good practice if you are going to read output file back and parse it to pick out only needed data. Here we just display the output file as is so this step can be omittted. The next step is a `PROC MEANS`. This is what we really want SAS to do for us. The last step is another `PROC PRINTTO`. This one has no options at all. It will simply redirect output back to SAS output window and log to SAS log window.

The full source code is listed in Appendix 1. Although it is a Delphi program, which use Object Pascal, it should be simple enough for any SAS programmer to understand it.

EASY INTERFACE FOR SAS (EI)

For a more realistic application, let's take a look at Easy Interface for SAS (EI). EI is a SAS/Assist type utility program developed mainly to help beginning SAS users. You interact only with the graphic user interface. SAS itself is hidden and

never visible to you. EI handles many details for you. For example, there is no need to know what a SAS library is since EI will create it automatically for you. You can also easily retrieve the generated SAS program, which will help you to learn SAS programming. If you already know SAS, you can also modify it and resubmit it as you can in the SAS editor.

The following is a list of functionality implemented in EI so far:

- File Manger
- SAS File Viewer (PROC CONTENTS, DATA Step)
- Data extraction (DATA Step)
- Sorting (PROC SORT)
- Import/Exporting from/to text files of different types
- Import/Exporting from/to Microsoft Excel Version 7 file
- Summary statistics (PROC MEANS)
- Frequency statistics (PROC FREQ)
- Correlation (PROC CORR)
- T-Test (PROC TTEST)
- Paired T-Test (PROC MEANS)
- ANOVA (PROC GLM)
- Regression (PROC GLM)
- Confidence interval for mean and for percentage
- Sample estimate for one and two sample
- Fast random sampling (DATA step)

The main screen of Easy Interface for SAS is a file manager shown in Appendix 2. SAS is invoked through Automation at the same time Easy Interface for SAS is started and stays invisible all the time. The user then clicks on the search button to get a list of directories containing SAS data sets in the left list box. The right list box shows SAS data sets in the selected directory. Double clicking on a SAS data set will display the contents of the data in a data set viewer, which looks like a spreadsheet. Basic file manger functions, such as copying, moving, deleting, and renaming files, and creating and deleting directories, are also implemented. In additional to SAS data sets, the user may select and display any SAS program. Other types of file filters can also be selected. In this case, double clicking on the file will open or run the file with the associated application. Several SAS procedures are available in EI. For example, Summary Statistics is an implementation of `PROC MEANS`. When invoked, a screen will appear (Appendix 2), displaying the name of the SAS data set, number of observation in the data set, a list of variables, their type (Numerical, Character, or Date), and their labels if available. The user then can use buttons or 'drag and drop' to select the variable(s) to be analyzed and to group variable(s). A list of statistics is available for users to select. By default, Mean, Sum, and Standard Deviation are checked. Clicking on the OK button will generate a SAS program and submit it to SAS through Automation. The result is displayed in the editor (Appendix 2). The user can also view the log file and the generated SAS program. Furthermore, the user may modify the generated SAS program and resubmit it within editor.

Now let's take a look at the implementation of `PROC MEANS` in EI. First, it retrieves information on the SAS data set. When the user selects Descriptive Statistics from menu, EI runs the following SAS program:

```
proc printto print= log='c:\temp\_SASFM.log';
options linesize=72 pagesize=56 nodate
        pageno=1;
libname _DATAIN "c:\sas\core\sashelp";
proc contents data=_DATAIN.salary
              out=_temp1 noprint;
data _null_ ;
set _temp1;
length T $ 1;
file 'c:\temp\_SASFM.lst';
if type=2 then t='C';
else if index(format,'YY'  )>0
     or index(format,'DATE')>0
then t='D';
else t='N';
put name $8. ' ' t $1. ' ' label $ 30. ' '
    nobs 10.;
run;
proc printto;
run;
```

The following output is in the text file `c:\temp\_SASFM.lst`

```
BEGDATE  D                               319
ENDDATE  D                               319
IDNUM    N Identification Number         319
JOBCODE  C                               319
SALARY   N Salary                        319
```

Note that Column 1 is the variable name. Column 2 is the type of Variable, with D for Date, N for Numeric, C for Character. Column 3 is the variable label, and the last column is the total number of observations in the data set.

This file is then read back by EI and the data is used in the input screen (Appendix 2).

In the input screen, the user sets up the analysis to be performed. It is necessary to validate user input. For example, at least one variable must be selected for analysis and it may not be a character variable. Input validation is often more difficult and time consuming to program than generating the SAS program itself. To minimize the required validation, EI, whenever possible, lets the user make selections with Check Boxes, List Boxes, Combo Boxes, etc. instead of typing input.

The following is an example of a SAS program generated by EI:

```
proc printto print='c:\temp\_SASFM.lst'
              log='c:\temp\_SASFM.log';
options linesize=72 pagesize=56 nodate
        pageno=1;
libname _DATAIN "d:\SAS\core\sample";
title 'Descriptive Statistics of salary';
proc means data=_DATAIN.salary
           Mean Sum STD maxdec=2;
var SALARY;
class JOBCODE; run;
```

```
proc printto;  run;
```

Note that the above SAS program is similar to the one used in the earlier demo application.

DEVELOPMENT PROCEDURE

For real business applications, it is recommended that developers use the following procedure:

- Develop and test SAS program(s) without any user interface or Automation.
- Convert SAS program into SAS macro with well-defined input, output, and any additional parameters.
- Design a Windows user interface that matches the SAS macro.
- Dynamically generate calls to the SAS macro and then submit it to SAS through Automation.
- Test, test, and test again.

This approach separates the SAS program from user interface development thus making debugging much easier. This also lets a SAS analyst team up with a Windows developer to create an application without much cross training. It is also a big advantage to application maintenance since one can modify the SAS macro to improve the analysis with minimal modifications to the user interface.

CONCLUSION

This paper presented an approach for using SAS Automation server as a database and analytic engine with a Windows user interface developed in Delphi, Visual Basic, or a similar tool. The technique is portable and generic enough to be used in diverse applications. Companies already using SAS as an analytic tool can use these methods as a fast means to turn SAS programs into Windows applications. Companies doing traditional application development can use these same techniques to tap into powerful SAS analytic procedures. In either case developers can continue using familiar development tools.

ACKNOWLEDGMENTS

Thanks to David Body of Big Creek Software for reviewing and editing this paper.

AUTHOR CONTACT

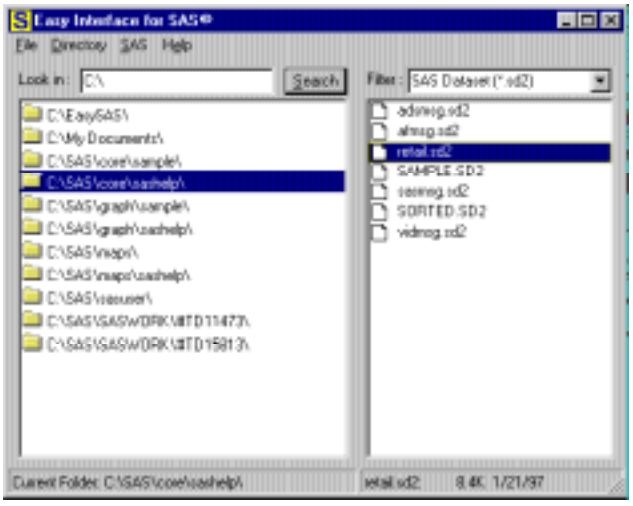
Zhuan (John) Xu
Big Creek Software, LLC
http://www.bigcreek.com
E-mail: johnxu@bigcreek.com

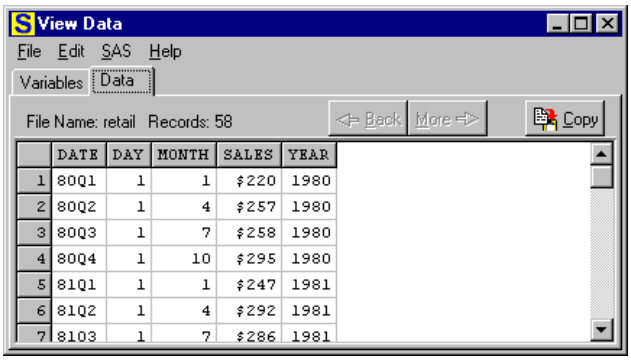
APPENDIX 1: DEPHI SOURCE CODE FOR DEMO

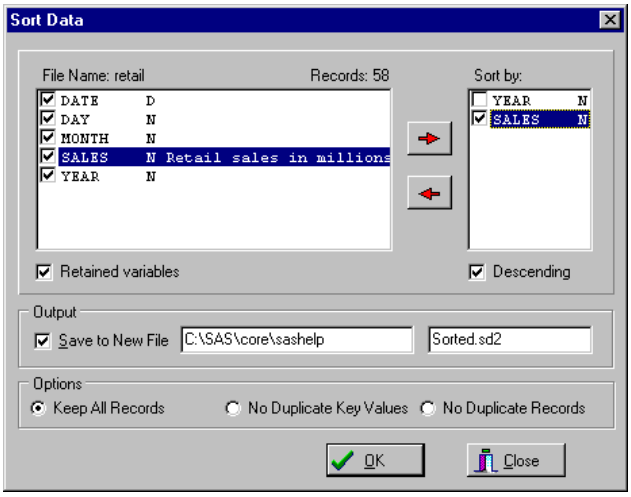
```
unit Unit1;
{
 Author: Zhuan (John) Xu/Big Creek Software

 Last Updated: December 29, 1997

 Prepared for SUGI27 presentation.
}

interface

uses
  Windows, Messages, SysUtils, Classes,
Graphics, Controls, Forms, Dialogs,
  StdCtrls, ComObj;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button3: TButton;
    Memo1: TMemo;
    Button4: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var
Action: TCloseAction);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  var SAS :  Variant;
implementation

{$R *.DFM}
procedure TForm1.Button1Click(Sender: TObject);
begin
  // Start SAS Server
  SAS := CreateOleObject('SAS.Application');

  SAS.Wait := True;
  //SAS.Visible := True;
  Button1.Caption := 'SAS Started';
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  // Make SAS visible/Invisible
  SAS.Visible := not SAS.Visible;
end;

procedure TForm1.Button2Click(Sender: TObject);
var SASProg: String;
begin
  // Detelete Output file
  if FileExists('c:\test\test.lst') then
     DeleteFile('c:\test\test.lst');

  // Create SAS program
  SASProg :=
    'PROC PRINTTO LOG=''c:\test\test.log'''
  + '           PRINT=''c:\test\test.lst'';'
  + #13+#10
  + 'LIBNAME demo ''c:\sas\core\sashelp'';'
  + #13+#10
  + 'OPTIONS LINESIZE=64 NOCENTER;'
```

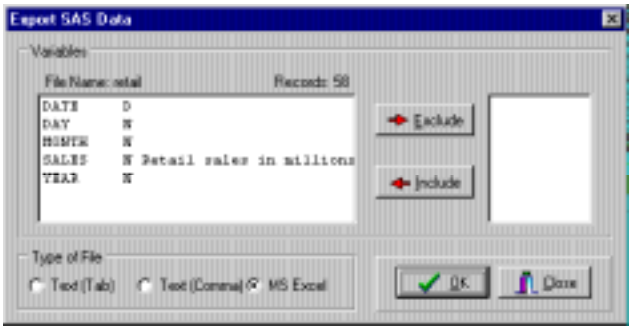
```
  +#13+#10
  + 'PROC MEANS DATA=demo.retail MAXDEC=2;'
  +#13+#10
  + '  VAR sales;'
  +#13+#10
  + 'PROC PRINTTO;RUN;';

  // Display Result
  memo1.lines.clear;
  memo1.lines.Add(SASProg);
end;

procedure TForm1.Button4Click(Sender: TObject);
var SASProg: String;
    i: Integer;
begin
  // Get SAS Program into SASProg
  for i:=0 to Memo1.Lines.Count-1 do
    SASProg := SASProg + Memo1.Lines[i]
               +#13+#10;

  // Submit Program
  SAS.Submit(SASProg);

  // Dispaly Result
  memo1.lines.clear;
  memo1.lines.loadfromfile('c:\test\test.lst');
end;

procedure TForm1.FormClose(Sender: TObject;
                       var Action: TCloseAction);
begin
  // Quit SAS Automation server
  SAS.Quit;
end;

end.
```

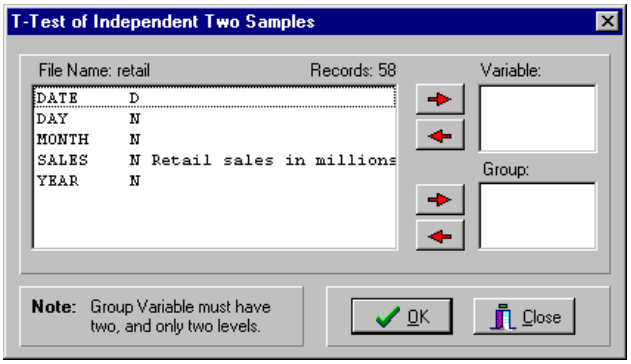
APPENDIX 2: SELECTED SCREENS OF EI (EASY INTERFACE FOR SAS)

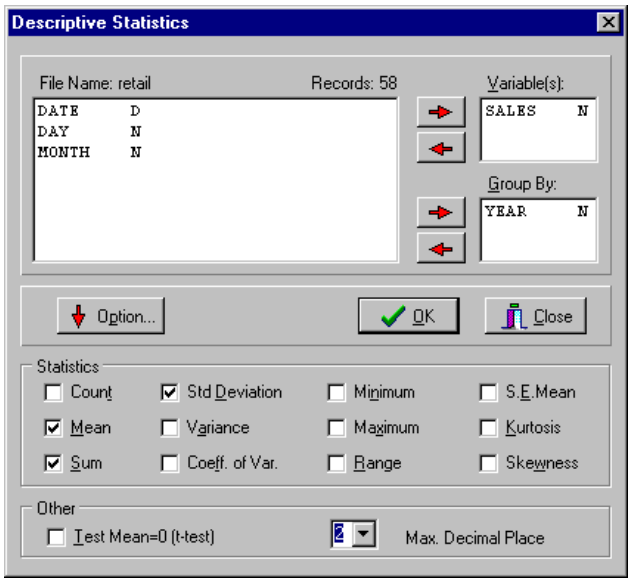


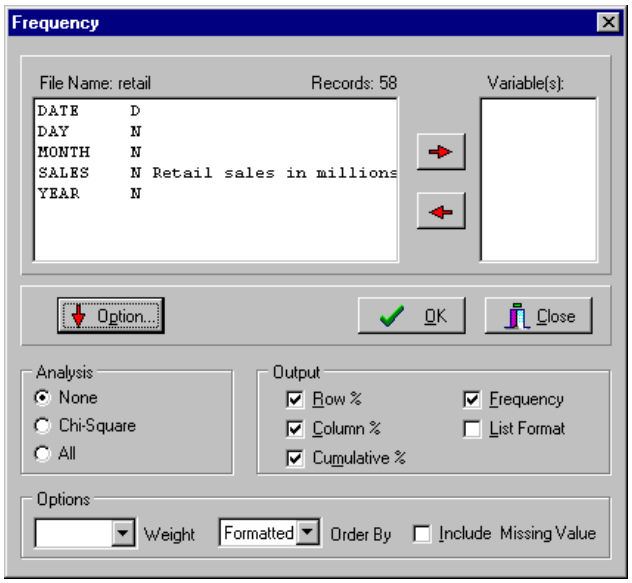


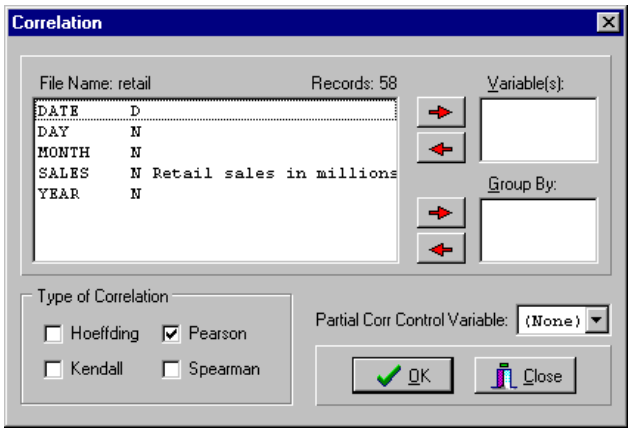


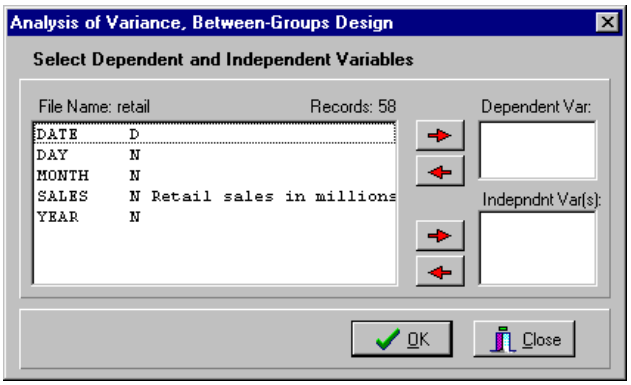


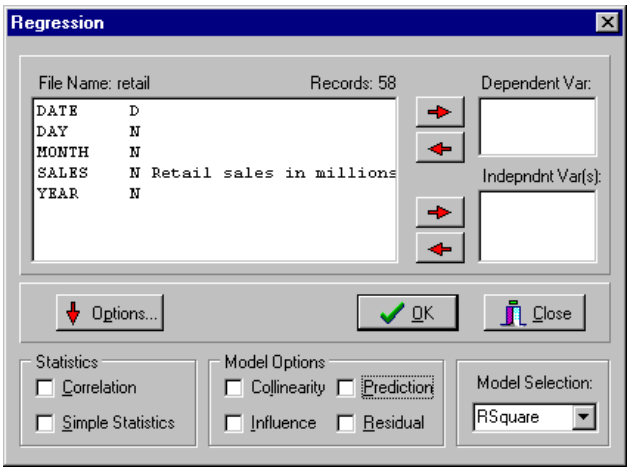


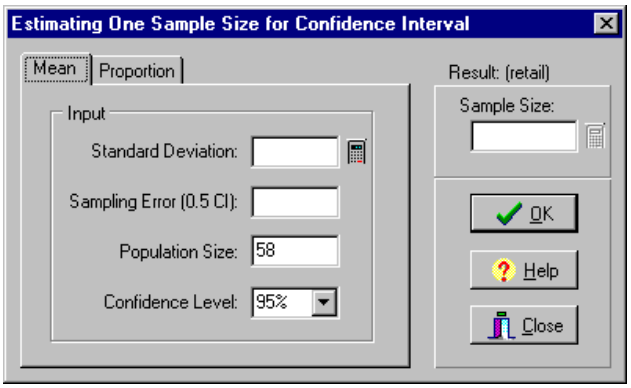

Descriptive Statistics
File Name: retail
Records: 58
DATE D
DAY N
MONTH N
Variable(s):
SALES N
Group By:
YEAR N
Option...
OK
Close
Statistics
Count
Std Deviation
Minimum
S.E.Mean
Mean
Variance
Maximum
Kurtosis
Sum
Coeff. of Var.
Range
Skewness
Other
Test Mean=0 (t-test)
2
Max. Decimal Place

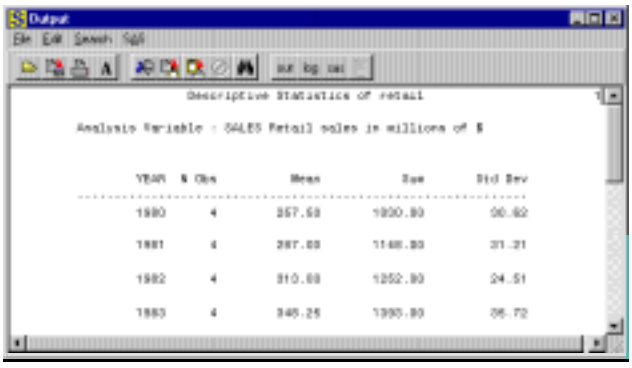

Frequency
File Name: retail
Records: 58
DATE D
DAY N
MONTH N
SALES N Retail sales in millions
YEAR N
Variable(s):
Option...
OK
Close
Analysis
None
Chi-Square
All
Output
Row %
Column %
Cumulative %
Frequency
List Format
Options
Weight
Formatted
Order By
Include Missing Value


Correlation
File Name: retail
Records: 58
DATE D
DAY N
MONTH N
SALES N Retail sales in millions
YEAR N
Variable(s):
Group By:
Type of Correlation
Hoeffding
Pearson
Kendall
Spearman
Partial Corr Control Variable:
(None)
OK
Close


Analysis of Variance, Between-Groups Design
Select Dependent and Independent Variables
File Name: retail
Records: 58
DATE D
DAY N
MONTH N
SALES N Retail sales in millions
YEAR N
Dependent Var:
Indepndnt Var(s):
OK
Close


Regression
File Name: retail
Records: 58
DATE D
DAY N
MONTH N
SALES N Retail sales in millions
YEAR N
Dependent Var:
Indepndnt Var(s):
Options...
OK
Close
Statistics
Correlation
Simple Statistics
Model Options
Collinearity
Prediction
Influence
Residual
Model Selection:
RSquare


Estimating One Sample Size for Confidence Interval
Mean
Proportion
Input
Standard Deviation:
Sampling Error (0.5 CI):
Population Size:
58
Confidence Level:
95%
Result: (retail)
Sample Size:
OK
Help
Close


Output
Descriptive Statistics of retail
Analysis Variable : SALES Retail sales in millions of $
YEAR N Obs Mean Sum Std Dev
1980 4 357.50 1430.00 60.62
1981 4 287.00 1148.00 31.21
1982 4 313.00 1252.00 24.51
1983 4 345.25 1381.00 86.72