PharmaSUG 2013 - Paper AD04

More Power to SAS - Embedding Other Programming Languages in SAS Through SAS/IML® Studio

Max Cherny, GlaxoSmithKline, King of Prussia, PA

ABSTRACT

Embedding code from different programming languages offers many new opportunities for SAS users to further enhance their SAS programs by taking full advantage of the features not yet available in SAS. SAS/IML Studio is a very powerful free SAS software which allows use of various programming languages together with SAS code. These languages include the IML programming language used for complex and fast data analysis, Java, C and R.

This paper explains the purpose and the basic functions of SAS/IML Studio. Easy and reusable examples of running SAS/IML, Java and R code in SAS/IML Studio will be provided. These examples can function either as the stand-alone programs or as the starting point for further integration of IML, Java or R programming languages into SAS code.

The paper will also provide side-by-side comparison of R and SAS syntax.

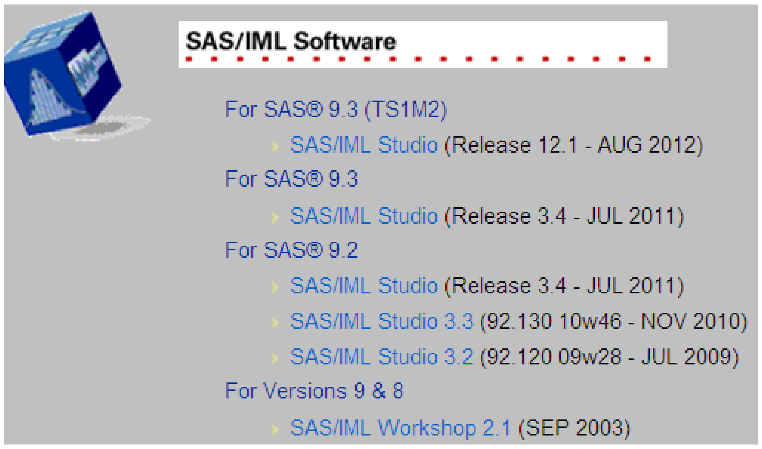
INTRODUCTION

The ability to use different programming languages together with SAS offers many new opportunities to SAS users. One way is to use SAS/IML Studio. SAS/IML Studio is a stand-alone product from the SAS Institute which is available to any SAS user. It can be downloaded from sas.com free of charge. It is a very powerful, relatively new application designed to perform a large variety of tasks. It allows SAS users to use IML matrix programming language together with SAS language.

SAS/IML Studio also allows SAS users to integrate Java, R or even C into their SAS programs. SAS users with the expertise in those programming languages can take full advantages of them to further enhance their SAS code with features not currently available in SAS. For example, Java or C code can help create powerful graphical user interfaces to use together with SAS language. Embedding R syntax allows SAS users to create powerful graphics using features not yet available in SAS. The latest R statistical routines not yet implemented in SAS can easily be used for exploratory data analysis.

INSTALLING AND RUNNING SAS/IML STUDIO

SAS/IML Studio is usually provided separately by SAS and has to be installed from sas.com. The link for installing the studio is http://support.sas.com/demosdownloads/setupcat.jsp?cat=SAS%2FIML+Software. Any licensed SAS user should be able to download the appropriate version of the application from that link.



SAS/IML Software

For SAS® 9.3 (TS1M2)
- SAS/IML Studio (Release 12.1 - AUG 2012)

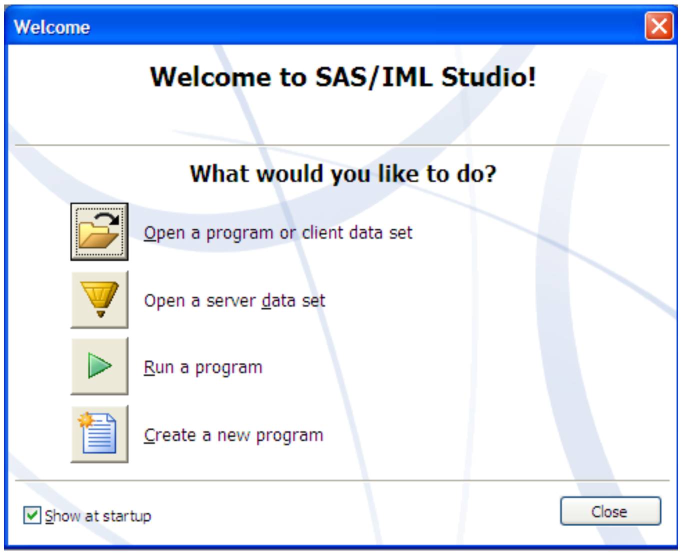
For SAS® 9.3
- SAS/IML Studio (Release 3.4 - JUL 2011)

For SAS® 9.2
- SAS/IML Studio (Release 3.4 - JUL 2011)
- SAS/IML Studio 3.3 (92.130 10w46 - NOV 2010)
- SAS/IML Studio 3.2 (92.120 09w28 - JUL 2009)

For Versions 9 & 8
- SAS/IML Workshop 2.1 (SEP 2003)

Display 1. Downloading SAS/IML Studio

After installing the application it should be easy to run it by clicking the desktop icon or running from the START menu. The initial dialog will give a number of options. It is recommended to use “Create a new program” option.


Welcome
Welcome to SAS/IML Studio!
What would you like to do?
Open a program or client data set
Open a server data set
Run a program
Create a new program
Show at startup
Close

Display 2. Starting SAS/IML Studio

SAS/IML Studio has a very large number of examples which should help anybody get started. The help facility of the studio is excellent and explains many complicated concepts in a very easy and straightforward manner.

Sometimes the initial run of SAS/IML Studio's examples may lead to various errors. Most errors should be fixed by the following steps:

1. Exit SAS/IML Studio
2. Open a Windows Command Prompt window.
3. In the Command Prompt window, use the CD command to change to C:\Program Files\SAS\SharedFiles\SGI\9.2. The directory may be different depending on the individual installation settings.
4. In the Command Prompt window, enter the following commands: *regsvr32 SGData.dll*, *regsvr32 SGGrid.ocx*, *regsvr32 SGPlot2D.ocx* and *regsvr32 SGRotate.ocx*.
5. Start SAS/IML Studio again

BASIC OPERATIONS IN IML PROGRAMMING LANGUAGE

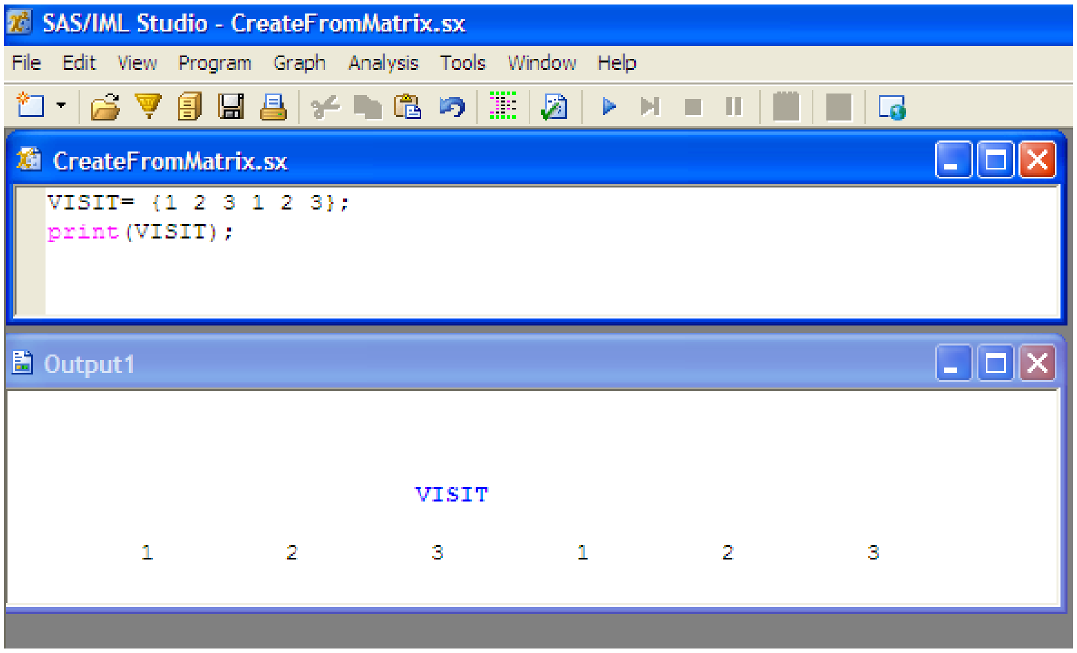
IML stands for Interactive Matrix Language and allows for various exploratory data analysis. It is a separate programming language from SAS. It has different syntax and routines. Its main advantage over SAS is the speed with which analysis can be done, as well as the fact that a number of statisticians or programmers are more familiar with the matrix programming. The IML language also provides a way to quickly create graphics.

The data matrix is a basic element of the language. While a matrix may look similar to a SAS dataset, it is actually a different data element.

A simple matrix VISIT is created by typing the following code:

```
visit = {1 2 3 1 2 3};
```

In order to execute this code one needs to highlight it and click the play button.


SAS/IML Studio - CreateFromMatrix.sx
File Edit View Program Graph Analysis Tools Window Help
CreateFromMatrix.sx
Output1

```
VISIT= {1 2 3 1 2 3};
print(VISIT);
```

```
                    VISIT

   1        2        3        1        2        3
```

Display 3. Running IML Statements

The VISIT matrix can be compared to VISIT column a SAS dataset produced by the following statement:

```
data vitals;
  input VISIT;
  datalines;
  1
  2
  3
  1
  2
  3
  ;
run;
```

Note that the matrix VISIT is not associated to any SAS-like dataset. Multilevel matrixes can also be defined using commas:

```
DiaBP = {67 70 71, 72 65 81 };
```

```
The DiaBP matrix looks as follows:
```

```
67          70          71
72          65          81
```

Various different operations and manipulations can be now applied to these matrixes. For example, it is possible to calculate the mean value of matrix elements using the syntax below:

```
meanDBP=  DiaBP[,:];
```

In order to see the resulting number, the PRINT function needs to be used:

```
print(meanDBP);
```

```
meanDBP

69.333333
72.666667
```

Output 1. Output of IML Print Statement

Various complex data and statistical analysis can be done using IML syntax in the similar manner. For example, a complex graph can be created in SAS/IML Studio. The following code will create a line plot of diastolic blood pressure by subject. The matrices will be linked to DataObject which is similar to a dataset data structure in SAS.

```
/* create matrixes containing BP values a subject 1 and 2 at visits 1, 2 and 3 */
VISIT = {1 2 3 1 2 3};

DiaBP = {67 70 71 72 65 81};

subjid = { "1" "1" "1" "2" "2" "2"  };

/* create dobj object */
declare DataObject dobj;
dobj = DataObject.Create( "Line Plot" );

/* Add matrixes to data objects */
dobj.AddVar( "Visits", VISIT);
dobj.AddVar( "DiaBP", DiaBP );
dobj.AddVar( "Subjid", subjid );

/* create line plot */
LinePlot.CreateWithGroup( dobj, "Visits", "DiaBP", "Subjid" );

/* additional code is required to add legend */
declare LinePlot plot;
plot = LinePlot.CreateWithGroup( dobj, "Visits", "DIABP", "Subjid" );

subjid= { "Subject 1" "Subject 2"  };
colors = RED // BLUE ;

run ColorCodeLinesByGroups( plot, "Subjid", colors );
run DrawLegend( plot, subjid, 12, colors, SOLID, NULL, 0xE8E8E8, "ORC" );
```

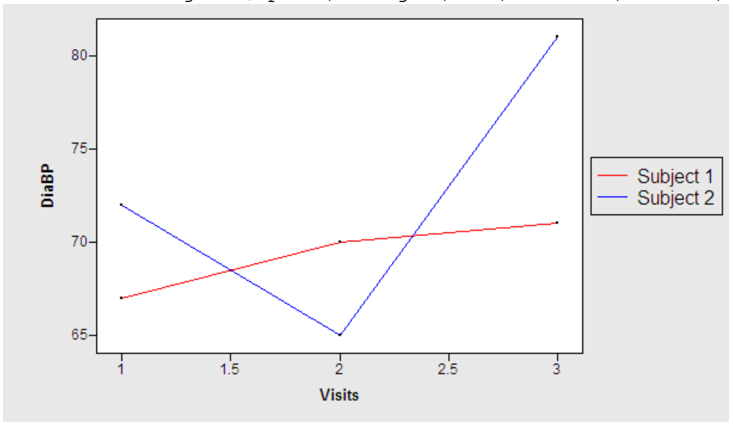


Figure 1. Basic Grouped Line Plot with Legend

RUNNING SAS CODE IN SAS/IML STUDIO

Since SAS/IML studio uses a different syntax from SAS; all SAS statements need to be inserted between SUBMIT and ENDSUBMIT statements. Regular SAS syntax can be used between these statements. The following simple SAS code will create a SAS dataset called TEST when run in SAS/IML studio:

```
submit; /* start of SAS code block */

data test;
do i=1 to 10;
 output ;
 end;
run;
```

```
   endsubmit; /* end of SAS code block */
```

Note that SAS code needs to follow the SUBMIT statement and end with ENDSUBMIT.

EMBEDDING JAVA WITHIN SAS CODE IN SAS/IML STUDIO

Java is a very popular programming language. It allows the creation of very powerful applications which can run on most platforms. Embedding Java syntax with SAS code can help SAS users with the knowledge of Java to create applications which cannot be designed by SAS alone. Any Java programming interfaces, such as GUI tools, will become available to be used together with the powerful data analysis tools of SAS.

In order to develop Java applications, the Java Development Kit (JDK) must be downloaded and installed in the local machine. JDK can be installed from http://www.oracle.com/technetwork/java/javase/downloads/index.html

Once JDK is installed, it is recommended to make sure it works correctly and basic Java code can be created in the appropriate directories. The JAVAC command will compile a Java source file and the JAVA command will run the application. JAVAC.exe can be usually found in a directory like C:\Program Files\Java\jdk1.7.0\bin. The exact path depends on specifics of the downloaded JDK module and the local machine's operating system.

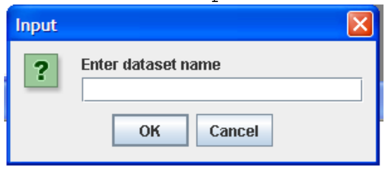
If the Java complier works, then it usually means it would work in SAS/IML studio environment as well. Java syntax can be inserted into SAS/IML studio as is. The following simple example will demonstrate a very basic use of Java as part of SAS/IML studio.

First, it is necessary to import all Java packages so that they will be available to the programs in the studio.

```
   import javax.swing.JOptionPane;
   import java.string.*;
```

The next statement is an example of Java message box asking for user's input. The input is the name of a dataset.

```
   declare String dataset; /* initialize variable for dataset */
   dataset = JOptionPane.showInputDialog("Enter dataset name", "");
```



Enter dataset name

Display 4. Simple Input Dialog Box Produced by Java syntax in SAS/IML STUDIO

This dialog box can be highly customized. The message box is now produced completely by Java programming language. This means that any functionality of Java can now be incorporated into SAS/IML studio and, consequently, used together with SAS code.

The code below will display the name of the dataset and print its contents.

```
   JOptionPane.showMessageDialog( null, "Displaying contents of " +dataset );

   submit dataset; /* pass name of the dataset entered */

     libname temp "C:\Documents and Settings\&sysuserid";

     proc contents data=temp.&dataset;

     run;

   endsubmit;
```

While this is a very simple code, it still demonstrates how Java programming language can be embedded into any SAS program to create sophisticated applications. An example would be a data entry application which allows users to enter various values into Java GUI and analyze those numbers using the power of SAS.

A different and more complicated approach is to compile Java classes from Java source code files and place those files into directories which the studio can access. This can be done by updating the location of class file in Option dialog menu.

C PROGRAMMING IN SAS/IML STUDIO

SAS/IML studio also lets SAS users who have expertise in C programming a way to combine functionality of C with the statistical and data analysis power of SAS. The approach is similar to that of using Java in SAS/IML studio.

RUNNING R IN SAS/IML STUDIO

One of the most powerful capabilities of the SAS/IML studio is its ability to embed R programming language as part of SAS programs. The R language is an open-source programming language for statistical computing. Due to its open-source nature and powerful graphic capabilities the popularity of R has grown over the past decade. Many statisticians and data analysts now use R for their clinical analyses in addition to SAS. In order to allow R to be used directly with SAS programs, SAS institute recently added R interface to SAS/IML studio.

It is very simple to run R in the studio. All that is necessary is to install R on the same machine which runs SAS. It is recommended to use the version of R supported by the available version of SAS/IML studio. Once R is installed, the SUBMIT/R statement is needed to be used before R code is invoked. ENDSUBMIT must follow the R code. Anything in between should be the regular R code.

```
Submit/R;
  plot(1:10)
endsubmit; /* end of R code block */
```

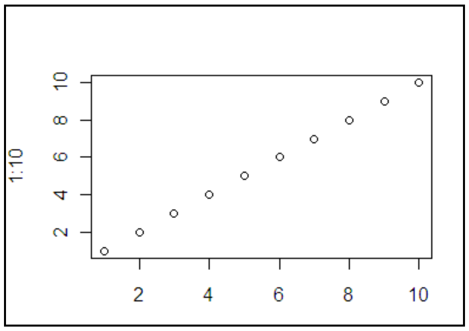


Figure 2. Basic R Line plot

Just like with IML or Java language, it would be helpful to learn the basic syntax of R. The table below provides code example of various operations in SAS and R. The left column contains SAS code examples, the right column shows the same code implemented in R along with some of the output. All the examples use VITALS dataset shown below.

Create VITALS dataset	

`data vitals;`
`  input subjid visit dia sys wt;`
`  datalines;`
`  1 1 67 113 170`
`  1 2 70 114 160`
`  1 3 71 120 200`
`  2 1 72 121 180`
`  2 2 65 111 140`
`  2 3 81 110 120`
`  ;`
`run;`
 	
`/* print contents of VITALS */`
`proc print data=vitals;`
`run;`
	`#standard R approach to create a dataset (called data frame in R)`
	`vitals= data.frame(`
	` subjid=c(1, 1, 1, 2, 2, 2),`
	` visit=c(1,2,3,1,2,3),`
	` dia=c(67, 70, 71 ,72 ,65 ,81),`
	` sys=c(113,114,120,121,111,110),`
	` wt=c(170, 160, 200, 180, 140, 120)`
	` )`
	
	`#different SAS-like approach :`
	`vitals=read.table(text= "subjid visit dia sys wt`
	` 1 1 67 113 170`
	` 1 2 70 114 160`
	` 1 3 71 120 200`
	` 2 1 72 121 180`
	` 2 2 65 111 140`
	` 2 3 81 110 120", header=TRUE, sep=""  )`
	
	`print(vitals)`
	`   subjid visit dia sys   wt`
	`1       1     1  67 113  170`
	`2       1     2  70 114  160`
	`3       1     3  71 120  166`

```
4       2      1  72 121  180
5       2      2  65 111  182
6       2      3  81 110 1812
```

Create a flag variable

```
data vitals;
 set vitals;
  length sys_flag $15;
  if sys > 115 then
sys_flag="High";
  else sys_flag="Low or Normal";
run;
```

```
vitals$sys_flag<-ifelse (vitals$sys>115, "High",
"Low or Normal")
```

Drop a column

```
data vitals;
  set vitals;
  drop sys;
run;
```

```
vitals$sys<-NULL
```

Rename a column

```
data vitals;
  set vitals;
  rename
dia=diastolic_blood_presure;
run;
```

```
vitals<-rename(vitals, c("dia" =
"diastolic_blood_presure"))
```

Calculate a new variable

```
data vitals;
  set vitals;
  wt_kg=wt/2.2;
run;
```

```
vitals$wt_kg<-vitals$wt/2.2
```

Copy a dataset

```
data vitals2;
  set vitals;
run;
```

```
vitals2<-vitals
```

Compare the datasets

```
/* make a copy of a dataset and
change some values */

data vitals2;
  set vitals;
  if _N_=4 then dia=81;
run;

/* compare datasets */

proc compare base=vitals
compare=vitals2;

run;
```

```
vitals2$dia[4]<-81

#compare datasets
all.equal(vitals, vitals2)
```

```
[1] "Component 3: Mean relative difference:
0.125"
```

Delete a dataset

```
proc datasets nolist;
  delete vitals;
quit;
```

```
rm(vitals)
```

Describe datasets

```
proc contents data=vitals;
run;
```

```
library(Hmisc)
contents(vitals)
```

Calculate maximum value

```
proc means data=vitals;
  var dia ;
run;
```

```
vitals$max_dia<-max(vitals$dia, na.rm = T)
```

Print number of rows in the data frame

```
proc sql noprint;
   select count(*) into: rows from
   vitals;
   quit;
%put Number of rows &rows;
```

```
number_of_rows=as.numeric(nrow(vitals))
print(number_of_rows)
```

Remove duplicate records

```
proc sort data=vitals
out=unique_vitals nodupkey;
   by subjid;
run;
```

```
unique_vitals<-subset(vitals,
!duplicated(subjid))
```

Subset data

```
data vitals_subjid2;
   set vitals;
   where subjid=2;
run;
```

```
vitals_subjid2<- subset(vitals,
  subjid==2 )
```

Merge two datasets

```
/* create DEMO dataset */
data demo;
     input subjid age;
     datalines;
     1 60
     2 55
     3 70
     ;
run;


/* merge DEMO with vitals */
proc sort data=demo;
     by subjid;
run;

proc sort data=vitals;
     by subjid;
run;

data vitals ;
  merge vitals (in=a)
           demo (in=b);
   by subjid;
   if a and b;
run;
```

```
#create demo data
demo= data.frame(
       subjid=c(1:3),
       age=c(60,55,70))

#merge vitals with demo
vitals<-merge(vitals, demo,
       by=c("subjid"), #merge by subjid
       sort=T) #sort by merging variables)

 print(vitals)
```

```
subjid visit dia sys   wt    age
1      1     1  67 113  170  60
2      1     2  70 114  160  60
3      1     3  71 120  166  60
4      2     1  72 121  180  55
5      2     2  65 111  182  55
6      2     3  81 110 1812  55
```

Transpose records

```
proc transpose
data=vitals(keep=subjid dia visit)
out=dia_transposed;
```

```
#need reshape library
library (reshape)
```

```
  by subjid;
  var dia;
run;
```

```
#keep only necessary variables
dia<-vitals[c("subjid", "visit","dia")]

#transpose
dia_transposed<-reshape(dia,
     idvar=c("subjid"),
     timevar="visit",
     direction="wide")

print(dia_transposed)
```

```
subjid dia.1 dia.2 dia.3
1       1     67     70     71
4       2     72     65     81
```

PROC IML

Many of the features of SAS/IML studio are also available in PROC IML. PROC IML can be directly called in SAS programs. For example, the following code will create VISIT matrix and print its contents as part of a SAS program.

```
proc iml;
  VISIT= { "1" "1" "1" "2" "2" "2"  };
  print (VISIT);
quit;
```

However, depending on the version of the SAS application, some functions of the studio may not be available. For example, R code can only be used in PROC IML starting with SAS 9.2 Ts2M3 Ship Event 10w46 or later. Additionally, GUI feature of the studio are not accessible through PROC IML.

CONCLUSION

SAS/IML studio offers SAS users a number of ways to add the power of different programming languages to SAS programs. SAS users with expertise in those languages may be able to dramatically enhance their SAS programs by using features of those languages not yet available in SAS.

Languages for use with SAS	**Key features**
Interactive Matrix Language	Matrix programming Fast statistical analysis
Java	Ability to create powerful platform-independent GUI applications
R	Open-source statistical language Strong graphical capabilities and statistical routines, many of which are not yet implemented in SAS. Popular in academia
C	One of the most powerful and popular programming languages
Fortran	Mainly a legacy programming language

Table 1. Languages available to SAS users through SAS/IML studio

REFERENCES

Shealy, D. Benefits of JAVA Within SAS: Exploring the Differences of the new JavaObj and SAS/IML studio, WUSS11, 2011. Available at http://www.wuss.org/proceedings11/Papers_Shealy_D_74867.pdf.

http://support.sas.com/rnd/app/studio/studio.html

http://www.r-project.org/

ACKNOWLEDGMENTS

Author would like to thank Greg Cicconetti, Ph.D. for his help with this paper.

CONTACT INFORMATION

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

Max Cherny
GlaxoSmithKline
Email: chernym@yahoo.com