SAS® and R Working Together

Matthew Cohen, Wharton Research Data Services

ABSTRACT

This paper will explore the ways SAS and R can work together. It will cover data transfers using SAS Transport and ASCII files and how to call R directly from within SAS. Rather than focusing on the pros and cons of each language, I will assume that some people want to use both.

INTRODUCTION

Below is a scenario of how someone might use both SAS and R. Even if this scenario does not apply to you, it's becoming more and more likely that you as a SAS programmer will be asked to provide data to someone using R or receive data from an R user to be integrated back into the SAS environment.

Both SAS and R can perform data management and create subsets. It appears to be more common to do this in SAS. One reason may be that SAS does not need to load the entire dataset into memory before creating the subset, and there may be other reasons as well. This paper will assume the data management is done in SAS.

Both SAS and R can run basic statistical procedures and create graphs. You could argue that one language is better than the other for this, or at least that specific procedures are better on the SAS or R side. But we know for sure that some people prefer to run graphs and/or statistics in R. For purposes of illustration, this paper will assume that the data will be transferred to R at this point: after the data processing is complete and the analysis is ready to start.

Since research is often an iterative process, this paper will assume that the results of the analysis in R must be transferred back to SAS.

Basics of R

This is not a paper on how to program in R, but there are some basic commands needed in order to know if the imports or exports work. Several graphical user interfaces are available including RStudio and RGUI. R can also be run in batch mode on a Windows or UNIX command line with the command

```
R CMD BATCH --no-restore rcode.R output_loglst.txt
```

Both the input program file and the output file are specified on the command line. R combines the equivalent of the SAS log and lst files into one.

R has multiple object types that store data including data frames, matrices, and lists. The R data frame is most similar to the SAS dataset, and this paper will focus on writing to and reading from data frames. R uses the terms 'columns' and 'rows' instead of 'variables' and 'observations', but this paper will use the SAS terminology. Case matters in R: for variable names, function names, and just about everything else. They are spelled correctly in the code examples, and are in upper case when otherwise referred to in the text, in keeping with SAS paper conventions.

The following statements can be used in both graphical and batch versions of R:

`names(dsname)`	Print the items in a list, or in this case the variables in the data frame.
`str(dsname)`	Print the internal structure of an R object: variable name, label, and first couple obs
`summary(dsname)`	Print more details than STR() including frequencies and means
`*describe(dsname)`	Similar to SUMMARY(), but more verbose
`dsname`	The object name itself prints its contents
`lm(var1~var2,data=dsname)`	Linear model of two variables in a data frame
`plot(dsname)`	Plot the data frame**

* DESCRIBE() is part of the HMISC package. It must be installed with the INSTALL.PACKAGES() function before its first use.
**To produce graphs in batch mode, X Windows must be running or an output device must be specified

```
pdf(file="~/r_files/graphics.pdf")
plot(dsname)
dev.off()
```

From SAS to R

ASCII Text Files

Using plain text, especially CSV (comma separated value), files works to connect just about any two data repositories. R and SAS are no exception.

In SAS:

```
proc export data=sashelp.class outfile="~/r_files/sasdata.txt" dbms=csv replace;
run;
```

in R:

```
dataframe = read.table("~/r_files/sasdata.txt", header=TRUE, sep=",")
```

Results:

```
> names(dataframe)
[1] "Name"   "Sex"    "Age"    "Height" "Weight"
> str(dataframe)
'data.frame':19 obs. of 5 variables:
 $ Name  : Factor w/ 19 levels "Alfred","Alice",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ Sex   : Factor w/ 2 levels "F","M": 2 1 1 1 2 2 1 1 2 2 ...
 $ Age   : int  14 13 13 14 14 12 12 15 13 12 ...
 $ Height: num  69 56.5 65.3 62.8 63.5 57.3 59.8 62.5 62.5 59 ...
 $ Weight: num  112 84 98 102 102 ...
> summary(dataframe)
      Name     Sex         Age            Height          Weight
 Alfred : 1   F: 9   Min.   :11.00   Min.   :51.30   Min.   : 50.50
 Alice  : 1   M:10   1st Qu.:12.00   1st Qu.:58.25   1st Qu.: 84.25
 Barbara: 1          Median :13.00   Median :62.80   Median : 99.50
 Carol  : 1          Mean   :13.32   Mean   :62.34   Mean   :100.03
 Henry  : 1          3rd Qu.:14.50   3rd Qu.:65.90   3rd Qu.:112.25
 James  : 1          Max.   :16.00   Max.   :72.00   Max.   :150.00
 (Other):13
```

R assigns the data types correctly for the most part. R has more detail than just character and numeric. Name is a “factor”, a categorical / class variable instead of regular character. Age is an integer, and height and weight are numeric. The EXPORT Procedure prints the variables names with the first letter in upper case, and they are read in this way by R. It is important to note that R variables are case sensitive, so they must be referenced exactly as they appear.

R functions include what would be procedures in SAS. As mentioned above, STR() and SUMMARY() combine elements of the CONTENTS, PRINT, and MEANS procedures. Functions in R are generally modified with positional parameters rather than options and statements.

SAS Transport Files

R can read SAS Transport files created with the EXPORT engine (.xpt extensions), but not Transport files created with the CPORT procedure.

In SAS:

```
libname rexport xport "~/r_files/sasdata.xpt";

data rexport.sasdata;
  set sashelp.class;
run;
```

In R:

```
library(Hmisc)
dataframe = sasxport.get("~/r_files/sasdata.xpt")
```

Results:

```
> names(dataframe)
[1] "name"   "sex"    "age"    "height" "weight"
> str(dataframe)
'data.frame':19 obs. of  5 variables:
 $ name  : chr  "Alfred" "Alice" "Barbara" "Carol" ...
 $ sex   : Factor w/ 2 levels "F","M": 2 1 1 1 2 2 1 1 2 2 ...
 $ age   : int  14 13 13 14 14 12 12 15 13 12 ...
 $ height: num  69 56.5 65.3 62.8 63.5 57.3 59.8 62.5 62.5 59 ...
 $ weight: num  112 84 98 102 102 ...
> summary(dataframe)
     name               sex         age            height          weight
 Length:19          F: 9   Min.   :11.00   Min.   :51.30   Min.   : 50.50
 Class :character   M:10   1st Qu.:12.00   1st Qu.:58.25   1st Qu.: 84.25
 Mode  :character          Median :13.00   Median :62.80   Median : 99.50
                           Mean   :13.32   Mean   :62.34   Mean   :100.03
                           3rd Qu.:14.50   3rd Qu.:65.90   3rd Qu.:112.25
                           Max.   :16.00   Max.   :72.00   Max.   :150.00
```

The differences between plain text files and SAS Transport files are minor. The variable names are in lower case, and the name variable is a character data type rather than a factor. This in turn changes how that variable is displayed in the SUMMARY() output.

This example used the SASXPORT.GET() function from the HMISC library. While not “built in” to base R, it is one of the more popular additions. Since R is an open source language, anyone can contribute additional libraries and functions. READ.SSD() from the FOREIGN library reads a SAS dataset and creates a SAS Transport file on the fly, then reads it in to R. It does this by writing a short SAS program, which it calls by itself.

```
library(foreign)
dataframe = read.ssd('~/r_files', "sasdata")
```

Optional additional parameters to write the xpt file, SAS log, and SAS lst upon failure.

```
,tmpXport="~/r_files/tmpssd.xpt",tmpProgLoc="~/r_files/tmpssd")
```

SAS code written by the R function to create the xpt file

```
option validvarname = v6; libname src2rd '~/r_files';
libname rd xport '~/r_files/tmpssd.xpt';
proc copy in=src2rd out=rd;
select sasdata;
```

The final result is the same as using SASXPORT.GET(), but the error checking included is very strict: processing terminates with any warning or error in the SAS log, even unimportant / unrelated warnings that occur during SAS startup.

READ.SAS7BDAT() in the SAS7BDAT library also exists, but currently only works with datasets created on Windows XP. Since I am using Windows 7, I could not assess it. There are probably many other functions available to do the same task, including two that I wrote myself: READSASBYLIBNAME() and READSASBYPATH().

```
dataframe=readSasByLibname(lib='sashelp', file='class')
dataframe=readSasByPath(lib='~/r_files', file=sasdata)
```

They are conceptually very similar to the existing R packages, but more flexible. The full code is included in the appendix.

SAS IML

The SAS IML Procedure can submit R statements from within SAS. It is not a version of R packaged with SAS, it uses the R system installed on the local operating system. The R_HOME environment variable must be set accordingly so SAS can find it. The –RLANG option must be included in the SAS invocation, and can't be turned on later as a security precaution. The status of this option can be checked with `proc options option=rlang; run;` SAS version 9.2 and IML Studio version 3.3 or higher are also required. Since the whole process operates within SAS, transferring data is simple.

```
proc iml;
run ExportDataSetToR("Sashelp.Class", "dataframe" );
submit / R;
  names(dataframe)
  str(dataframe)
  summary(dataframe)
endsubmit;
quit;
```

There are 3 methods of transferring SAS data to R

Table 11.1 Transferring from a SAS Source to an R Destination

Method or Module	SAS Source	R Destination
ExportDataSetToR	SAS data set	R data frame
ExportMatrixToR	SAS/IML matrix	R matrix
DataObject.ExportToR	DataObject	R data frame

http://support.sas.com/documentation/cdl/en/imlsstat/63545/HTML/default/viewer.htm#imlsstat_statr_sect004.htm

This is the easiest way of working with SAS and R together, with a couple small caveats. In batch mode, R output is written to the lst file, but certain notes about R are written to the screen (standard output), which is not the most useful. When running R directly, the code is written to the log just before the corresponding output, similar to what the MPRINT option does in SAS. However in IML, SAS prints the R output without the corresponding code. In a long program it can be difficult to follow.

Variable Labels

In general, R does not support variable labels, however some of the functions in the HMISC library such as STR() and DESCRIBE() will display them. Notice that the four variables with labels no longer have the correct data type reported in STR(), they have taken on a new data type of Class 'labelled'. DESCRIBE() does not report data types, so there is no change.

```
> str(dataframe)
'data.frame':19 obs. of  5 variables:
 $ name  :Class 'labelled'  atomic [1:19] Alfred Alice Barbara Carol ...
  .. ..- attr(*, "label")= chr "First Name"
 $ sex   : Factor w/ 2 levels "F","M": 2 1 1 1 2 2 1 1 2 2 ...
 $ age   :Class 'labelled'  atomic [1:19] 14 13 13 14 14 12 12 15 13 12 ...
  .. ..- attr(*, "label")= chr "Age in years"
 $ height:Class 'labelled'  atomic [1:19] 69 56.5 65.3 62.8 63.5 57.3 59.8 62.5 62.5
  59 ...  .. ..- attr(*, "label")= chr "Height in inches"
 $ weight:Class 'labelled'  atomic [1:19] 112 84 98 102 102 ...
  .. ..- attr(*, "label")= chr "Weight in pounds"
> describe(dataframe)
dataframe

 5  Variables      19  Observations
--------------------------------------------------------------------------------
name : First Name
      n missing  unique
     19       0      19
```

The output above used SAS Transport files to transfer the data, text files and IML do not transfer variable labels. This is only a minor advantage for Transport files because the labels are not used widely in R anyway.

Missing Values

R uses NA to mark missing values in both character and numeric variables.

```
data sasdata;
  set sashelp.class;
  if _n_ = 3 or _n_ = 4 then do;
      sex = "";
      age=.;
      height=.;
  end;
run;
```

R imports the numeric missing values correctly, but treats the character as an empty string rather than a missing value. Both CSV and Transport files produce the same results, while IML transfers all three variables correctly.

```
> summary(dataframe)
Sex          Age             Height
 : 2    Min.   :11.00   Min.   :51.30
F: 7    1st Qu.:12.00   1st Qu.:57.50
M:10    Median :13.00   Median :62.50
        Mean   :13.29   Mean   :62.14
        3rd Qu.:15.00   3rd Qu.:66.50
        Max.   :16.00   Max.   :72.00
        NA's   :2       NA's   :2
```

The function IS.NA() is used to test for missing values. `is.na(dataframe)` will print true or false for each value in the whole data frame and `is.na(dataframe$age)` will test each value of the age variable for missing and print true or false.

Formats

Using the sashelp.citiday dataset, the following date formats were tested to see how they would be treated in R: date9, yymmddn8, yymmddd10, yymmdds10, mmddyyd10, and mmddyys10. The label was cleared from the date variable to give a more accurate reading of the results in R. For example

```
data sasdata;
  set sashelp.citiday;
  label date = ;
  format date date9.;
run;
```

In text files, most of the formats are read in as factors, meaning they are basically treated as text. Changing data types in R is simple, one area in which SAS could use improvement. AS.DATE() converts variables to dates and accepts the FORMAT= argument (which is also a standalone function). Together they act like a date INFORMAT in SAS. While there are fewer options in R than date INFORMATs in SAS, they cover most cases.

Code	Value	Example
%d	Day of the month (decimal number)	01-31
%a	Weekday (abbreviated)	Mon
%A	Weekday (unabbreviated)	Monday
%m	Month (decimal number)	01-12
%b	Month (abbreviated)	Jan
%B	Month (unabbreviated)	January

%y	Year (2 digit)	12
%Y	Year (4 digit)	2012

To import date9

```
dataframe$DATE=as.Date(dataframe$DATE, format="%d%b%Y")
```

To import yymmdds10

```
dataframe$DATE=as.Date(dataframe$DATE, format="%Y/%m/%d")
```

Since there are no DATA steps in R, the function names the data frame to which the variable belongs. In sashelp.citiday, the date variable is in uppercase. It is written to the CSV file and imported to R this way, and so it must be referenced in uppercase. In addition to simple, built-in dates, there are also CHRON and POSIX classes and many functions in the LUBRIDATE package that have much more flexibility and complexity. The yymmddn8 format could not be read in with AS.DATE() and might need one of the more advanced functions.

For SAS Transport files, the date9 format is converted directly to an R date. When using the other formats, the values are converted to their numeric representation, days since January 1, 1970 (compared to January 1, 1960 in SAS). The R statement below converts the values to dates and gives them the default date format in R (yyyy-mm-dd).

```
class(date) = c('POSIXt','POSIXct')
```

SAS IML converts date variables correctly regardless of the format.

From R to SAS

ASCII Text Files

The WRITE.FOREIGN() function in the FOREIGN library writes a text file along with the SAS program to read it in. Character variables are encoded in the dataset to reduce size and are written as formats in the program to recreate them.

```
library(foreign)
write.foreign(dataframe,"~/r_files/rdata_class.txt","~/r_files/rdata_class.sas",
package="SAS")
```

SAS Transport Files

The WRITE.EXPORT() function in the SASXPORT library works in a similar way. It creates an xpt file that has a data table and a format table, but the user has to do the work to reconstruct it. Also, SAS does not recognize the data table after import, although you can see both in the XPT file using the Universal Viewer.

```
library(SASxport)
write.xport(dataframe, file="~/r_files/rdata_class.xpt")
```

SAS IML

Reading data from R in SAS IML works the same way as writing data to R. Four functions are available.

Table 11.2 Transferring from an R Source to a SAS Destination

Method or Module	R Source	SAS Destination
DataObject.AddVarFromR	R expression	DataObject variable
DataObject.CreateFromR	R expression	DataObject
ImportDataSetFromR	R expression	SAS data set
ImportMatrixFromR	R expression	SAS/IML matrix

http://support.sas.com/documentation/cdl/en/imlsstat/63545/HTML/default/viewer.htm#imlsstat_statr_sect005.htm

CONCLUSION

SAS and R work well together and it is fairly easy to transfer data back and forth. If the objective is to use both languages together, IML is the easiest way to do so. When using R outside of SAS, both text and SAS Transport files work well, with the Transport files having a couple small advantages.

Since it is easy enough to use both languages together, it creates opportunities to use the strengths of each. SAS has more database-like capabilities including views, indexes, and streaming data into memory as needed. On the other hand, Revolution Analytics maintains a proprietary version of R and they claim to have solved the in-memory limitations of R. Others are working on this functionality for the open source version of R. It can also be useful to have the entire dataset in memory when doing analysis. Looking ahead and behind the current row (leads and lags) is a complicated task in SAS, but is trivial in R. Changing data types simply and having sophisticated logic for variable selection are also nice features.

One of the biggest differences to consider is the fact that R is open source. This adds flexibility and decreases the time needed to add new features. However, it introduces question of quality control. Different contributors have different skill levels and may spend different amounts of time testing. But the code is open for everyone to see, making it easier to find and fix problems.

There is no need to choose, use both!

RESOURCES

There are several great resources for SAS programmers working with R

For an overview of the language and software downloads: www.r-project.org
For details and syntax of functions: in RGui submit `help(function_name)`
A guide to getting started in R for SAS programmers: www.r4stats.com
An in-depth look at R through IML: http://www.analytical-software.de/fileadmin/doc/papers/HMS_-_PhUSE_2011_-_Calling_R-Functions_from_SAS.pdf

ACKNOWLEDGEMENTS

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.

CONTACT INFORMATION

Matthew Cohen
IT Director
Wharton Research Data Services
cohenmw@wharton.upenn.edu
http://wrds.wharton.upenn.edu

SAS programs to create SAS transport files

By Libname

```
/*parse sysparm command line arguments*/
data _null_;
  length sysparm current param value $ 200;
  sysparm = symget('sysparm');
  do i=1 to countw(%str(sysparm), ',');
    current = left(scan(sysparm, i, ','));       /* name=value */
    param = left(upcase(scan(current, 1, '='))); /* name */
    value = left(scan(current, 2, '='));         /*value */
    valid = not verify(substr(param, 1, 1),'ABCDEFGHIJKLMNOPQRSTUVWXYZ_')
    and not verify(trim(param),'ABCDEFGHIJKLMNOPQRSTUVWXYZ_0123456789')
    and length(param) <=32; /* Ensure valid V8 macrovar name */
    if valid then call symput(param, trim(left(value)));
  end;
run;

libname rexport xport "./sasdata.xpt";

data rexport.sasdata;
  set &lib..&file;
run;
*******************************************************************************
```

By Path

```
data _null_;
  length sysparm current param value $ 200;
  sysparm = symget('sysparm');
  do i=1 to countw(%str(sysparm), ',');
    current = left(scan(sysparm, i, ','));       /* name=value */
    param = left(upcase(scan(current, 1, '='))); /* name */
    value = left(scan(current, 2, '='));         /*value */
    call symput(param, trim(left(value)));
  end;
run;

libname rexport xport "./sasdata.xpt";
libname saslib "&lib";

data rexport.sasdata;
  set saslib.&file;
  if _n_ <= 10;
run;
```

R programs to call the SAS programs above and read SAS Transport files

By Libname

```
readSasByLibname = function(lib, file) {
  #pass command line parameters (lib,file) to SAS program sasxpt
  sas_str=paste("sas sasxptByLibname.sas -sysparm 'lib=",lib,",file=",file,"'",sep="")
  rc=system(sas_str)

  #if SAS extraction has an error then quit
  if (rc > 1) {warning("SAS dataset does not exist or is not readable.  Exiting R")
               quit(save="no", status=10, runLast=TRUE)
              }

  #Read SAS Transport file (xpt format) created by sasxpt
  library(Hmisc)
  dataframe = sasxport.get("~/r_files/sasdata.xpt")
  #rc=system("rm sasdata.xpt")

  return(dataframe)
}
dataframe=readSasByLibname(lib='sashelp', file='class')

*******************************************************************************
```

By Path

```
readSasByPath = function(lib, file) {
  #pass command line parameters (lib,file) to SAS program sasxpt
  sas_str=paste("sas sasxptByPath.sas -sysparm 'lib=",lib,",file=",file,"'",sep="")
  rc=system(sas_str)

  #if SAS extraction has an error then quit
  if (rc > 1) {warning("SAS dataset does not exist or is not readable.  Exiting R")
               quit(save="no", status=10, runLast=TRUE)
              }

  #Read SAS Transport file (xpt format) created by sasxpt
  library(Hmisc)
  dataframe = sasxport.get("~/r_files/sasdata.xpt")
  #rc=system("rm sasdata.xpt")

  return(dataframe)
}
#dataframe=readSasByPath(lib='~/r_files', file='testds')
```