Paper AD08

Peaceful Co-existence: R and SAS

Martin Gregory, Merck
Manuel Cornes, Merck

1 Abstract

R and SAS are very different programming environments. Not only are there features differences, but the radically different approach means that tasks which are difficult to implement in one environment may be simple in the other. Transferring data between the two environments is not trivial but both export and import is possible. There are also situations where using both languages for different parts of a task may be an efficient approach. Interaction between the two is not, however, trivial. Apart from a limited feature in SAS/IML, there is no easy way to call one from the other and seamlessly transfer inputs and outputs. In this paper we discuss the problems and propose solutions based on the specific task of building a bookmarked PDF from individual tables, listings and figures.

2 Introduction

We were requested to provide a program which combines a collection of text files containing tables, listings and pointers to image files (pdf, eps) containing figures into a single pdf file with hierarchical bookmarks derived from the titles of the tables, listings and figures. The program should be available to both SAS and R users. Input to the program should be a list of files to combine. These files may contain one or more multi-page tables or listings or instructions on which image file to insert. The titles have a specific structure to allow the program to automatically determine the start of a new table, listing or figure and to compute the bookmarks. For figures, we require a text file, or page within a text file, containing the titles and the path to the actual graphic file.

Given that we also generally need to make results, especially data, from one language in the other, we investigated how we could use both in tandem.

3 Implementation choices

For the implementation we decided for resource reasons and for ease of maintenance to implement in either SAS or R and to provide the program to users of the other language through a wrapper function. We chose R for a number of reasons. First is the relative ease with which R can interface with the operating system in a transparent manner [1]. Second is that the *pdf* graphics device and the *text* function, both components of the core R distribution, provide a fine-grained way to control writing text to a pdf file. Third, the calculations required for determining layout parameters, page sizes and titles for use in bookmarks and for checking the consistency of the output require a complex, nested data structure and R's LISP heritage provides easy access to arbitrarily complex data structures using nested lists. Finally, building PDF can be a resource intensive process, a lesson we learned with an earlier implementation done using SAS and Ghostscript. Calculating bookmarks and checking for consistency requires multiple passes of the input files which is avoided using R as it holds the data in memory.

The actual creation of the pdf file is done with R for text portions (i.e., tables and listings). When figures are provided as eps files, conversion to pdf is done with ghostscript [2]. Finally, pdftk [3] is used to concatenate the resulting pdfs and add bookmarks .

So we developed an R function *lst2pdf* and a wrapper SAS macro *%lst2pdf* which calls the R function. The syntax of both function and macro are very similar:

```
value <- lst2pdf(input=, output=, regulatory=FALSE, ## R
         watermark="DRAFT", gs="gs", pdftk="pdftk", debug=FALSE)

%lst2pdf(input=, intype=LIST|FILE, output=, regulatory=FALSE, /* SAS */
         watermark=DRAFT, gs=gs, pdftk=pdftk, debug=FALSE, rc=rc_lst2pdf) ;
```

The first difference is that we need an additional parameter, *intype*, to tell SAS whether the value of the *input* parameter is a file name or the name of a dataset containing a list of files. This is not necessary for the R function because the *input* parameter is always a character vector - a single input file means that the vector has only one element. The second difference is that the return code from the R function is the value returned by the function on so the *rc* parameter is not required.

The SAS wrapper macro does three tasks: it transfers the dataset containing the list of files to process to an R data frame; it generates and executes an R program which calls the *lst2pdf* function with the appropriate parameters; and it reports on the outcome. Our particular example is fairly simple as far as the interaction is concerned but we can use it to illustrate the main points.

4 Transferring data

The first issue we have to deal with is making data in one system available to the other. Both SAS and R provide structures for tabular arrays of data similar to relational database tables: the SAS dataset and R data frame, respectively. R can additionally manipulate and store abitrary data structures using the list concept for which there is no corresponding functionality in SAS which can be used outside SCL[1].

4.1 Tabular data

R provides functions for converting data frames to SAS datasets and vice versa. If the SAS/IML module is available, SAS can transfer datasets to dataframes and vice versa. Another possibility is to use a DBMS for which both R and SAS have adapters.

There are, however, some limitations. The first is that while SAS has only two data types, floating point and character, R has a larger set of basic types: logical, integer, double, complex, character and raw. There are also many "complex" types extending these basic types, of particular interest are factor, date, and POSIXct[2], so conversions from R to SAS may involve changing the type. The functions for conversion are described in the following section.

4.1.1 foreign::read.xport

The *read.xport* function from the *foreign* package can read the public domain SAS V5 transport format datasets. This means that any data satisfying the rules for CDISC data models can be converted.

[1] data step hashes can potentially manipulate nested lists but storing them outside the data step would require writing a non-trivial serialisation routine.

[2] UCT date and time with origin at the UNIX epoch 1970-01-01T00-00-00

4.1.2 foreign::read.ssd

The *read.ssd* function from the *foreign* package can read any SAS V6, V8 or V9 dataset once the following conditions are satisfied:

1. SAS is available on the same host on which R is executing;
2. the dataset satisfies the rules for SAS V5 transport files;
3. for V6 datasets, the datasets were created on the same operating system as that on which R is executing

The function works by generating a SAS program to convert the datasets to the SAS V5 transport format and then uses the *read.xport* function to convert to data frames. For CDISC this is sufficient, but cannot be used if the datasets do not satisfy the constraints. Determining that this is the case is tricky as the function simply reports that there was a problem. Furthermore even a warning message in the SAS log will cause the function to terminate abnormally and not return the data frame and this is not reported in the R transcript.

4.1.3 foreign::write.foreign

The *write.foreign* function from the *foreign* package can convert data frames to SAS[3] datasets by writing a comma-separated file and a SAS program to read the file and create a SAS dataset. In our particular example this is not required, but we still discuss it as it may be relevant for other tasks.

Integer, double, logical, character basic types and the complex types factor, date and time (Date, dates, date, and POSIXt classes) are supported. For factors, the function creates a numeric variable and an associated format with the character values. The *write.foreign* function expects arguments which define the location of the comma-separated data file, the generated SAS code and the path for the library in which to write the SAS dataset. The function assumes that both R and SAS are using the same physical path for the data file. Since *write.foreign* does not require SAS and R to be on the same system, minimal post processing in the R program calling the function can adjust this path. Let us consider the following call to the function, assuming that the data and code files should be written to the directory *transfer* in the currrent working directory:

```
write.foreign(books,
              datafile="transfer/books.dat",
              codefile="transfer/books.sas",
              package="SAS",dataname="books",validvarname="V7",
              libpath="/data/project/sasdata")
```

Let us further assume that the data and code files will be moved to a different host where SAS is installed and copied to the */data/project* directory. The we can use this code to adjust the path:

```
books.f <- file("transfer/books.sas",open="r")
books.c <- readLines(books.f,n=-1,TRUE)
close(books.f)
books.c <- sub('INFILE  "transfer/','INFILE  "/data/project/',books.c)
books.f <- file("transfer/books.sas",open="w+")
writeLines(books.c,books.f)
close(books.f)
```

[3]SPSS and STATA are also supported

4.1.4 DBMS

Both SAS and R are capable of accessing DBMS tables, so one option is to avoid transfer completely and store data in a relational database. While at first glance this appears to be a simple solution, the question of conversion of data types arises again.

In the case of R and MySQL, for example, the MySQL types *date*, *time* and *timestamp* are converted to character. While it is trivial to convert these character values to the corresponding R types, it means that a table cannot be used directly. For SAS, how conversion is done

Table 1: How MySQL types map to SAS and R

MySQL	SAS	R	MySQL	SAS	R
bigint	numeric	double	datetime	numeric	character
mediumint	numeric	integer	timestamp	numeric	character
smallint	numeric	integer	time	numeric	character
tinyint	numeric	integer	year	numeric	character
decimal	numeric	double	char	character	character
int	numeric	integer	varchar	character	character
double	numeric	double	binary	character	character
float	numeric	double	varbinary	character	character
date	character	numeric	enum	character	character
set	character	character			

depends on whether the table is accessed via a libname engine, the *access* procedure or the *dbload* procedure. For MySQL the full extent of conversion to both SAS and R is shown in Table 1. While conversion may not automatically choose the correct type, it does generally provide a representation that can be converted in the target system.

For some databases, SAS does not support some data types at all. For details see the SAS/ACCESS manual [4]

4.2 Non-tabular data

Both systems also generate collections of tables. Most SAS statistics procedure can produce a collection of datasets containing various information from the analysis using the *ODS OUTPUT* statement. R list structures associated with its concept of class allows a more flexible approach. For example the object produced by a call to the *survfit* function is an object, actually a list, of type *survfit* containing 14 elements describing the number of subjects in each group, observation time, number at risk, whether the subject experienced an event, whether the subject was censored, censoring type, strata, survival function, standard error, upper and lower confidence intervals, CI type and percentage and, finally the call to the function. This information cannot be stored in a single table because the number of rows differ, but a list does not require all elements to be of the same dimension. There is no easy way to transfer these types of data structures.

5 Calling R from SAS

In this section we describe the options available for calling R from SAS and how to handle return status. There are at least four different ways of doing this:

SAS/IML Using the *submit /R* statement R code can be submitted from the *IML* procedure. Futhermore, the call routines *ExportDatasetToR* and *ImportDatasetFromR* can transfer

datasets to and from R. In SAS 9.3, we were able to run R code but not transfer datasets. In order to have the feature work correctly it is necessary to compile R with support for shared libraries which, according to the Configuration Options section of the R Installation and Administration Guide [5] may impose restrictions on using additional packages containing C code, cause symbol conflicts and impose an architecture dependent performance penalty of between 10 and 20 percent;

pipe file reference This allows a data step to read from the pipe and retrieve the messages returned by the R program execution (the R transcript). It does not provide a return code;

system function This data step function allows us to retrieve the return code but does not directly return the R transcript which needs to be read separately;

x statement The *x* statement neither retrieves the return code nor the transcript.

Generally we need either the return code or the transcript or both, so the pipe and system functions are the best candidates.

6 Calling SAS from R

R provides three different ways to execute an external command:

system() executes a command. If the argument *intern* is TRUE the function returns the exit status of the command. If the argument *intern* is FALSE it returns the standard output and standard error as a character vector. This function has some issues on Microsoft Windows;

system2() is a replacement for *system* which works well on both Microsoft Windows and UNIX-alike systems;

pipe() provides a connection which the program may write to or read from. This behaves in a similar way to the pipe file reference in SAS

We have not used these functions in a real application.

7 The lst2pdf solution

For our specific application the approach we chose is to first pass the input dataset by writing a comma-separated file and an R program to read the file into a data frame, append a call to the *lst2pdf* function to the generated R program and then execute the R code using a pipe.

7.1 Passing the input dataset from *%lst2pdf* to *lst2pdf*

Since the list of input files includes the full path, the constraint of 200 characters for SAS V5 transport files rules out using the *read.xport* and *read.ssd* function. One of the authors has written a SAS macro, *%sasds2df* and an R function which can convert any SAS dataset to an R data frame, provided both SAS and R are available. The macro generates a comma-separated file which includes the necessary metadata for type conversion and the R function uses this metadata to do the conversion[4]. For the *sashelp.class* dataset this would be:

[4]We plan to publish the macros and R function in the near future

```
'character','character","numeric","numeric","numeric"
"Name","Sex","Age","Height","Weight"
"Alfred","M",14,69,112.5
"Alice","F",13,56.5,84
...
"William","M",15,66.5,112
```

7.2 Executing the R code

%lst2pdf appends a call to the *lst2pdf* function to the code generated by *%sasds2df*. It also generates R code to write the return code to the transcript so that it can be parsed when executing the pipe, overcoming the fact that the pipe does not provide a return code. The R code is executed using standard I/O:

```
filename r1 pipe "R &rdebug --no-save --no-restore <&rpgm" ;
```

to avoid writing the transcript to a file. This allows us to use a single data step to execute the pipe and parse the transcript and insert it directly into the SAS log so that the entire "log" is in a single file. The *&rdebug* will contain debugging options for R if the parameter *debug=TRUE* was passed. The *–no-save* and *–no-restore* options prevent unintended loading or saving of a session *.RData* file. The actual data step used to execute the pipe is as follows:

```
01  data _null_ ;
02   length rc $8 ;
03   do until (eof) ;
04      infile r1 end=eof ;
05      input ;
06
07      if  _infile_ = "WARNING: ignoring environment value of R_HOME"
08         then continue ;
09
10      if scan(_infile_,1,":")="R_TO_SAS_RETURN_CODE" then do ;
11         %if &debug=Y %then %str(put _infile_ ;);
12         rc=strip(scan(_infile_,2,":")) ;
13         call symput("&rc",rc) ;
14         end ;
15      else put _infile_ ;
16      end ;
17   stop ;
18  run ;
```

Lines 07-08 are necessary as our installation writes this warning message at the start of the transcript and we don't want it picked up as a warning by the code which post-processes the

SAS log. Lines 10-14 show how the return code from *lst2pdf* is retrieved - the R function writes an identifiable string to the transcript and this is parsed by the data step. All other lines in the transcript are written to the SAS log.

8 Conclusion

For transferring data we could not use the built-in features as our dataset variable may exceed the 200 character limit. For the specific application we could simply have written a text file from SAS and read it from R, but as we have a need to transfer more general datasets for future applications, we chose to develop a more generic solution. In general, as long as we are dealing with integer, floating point or character data types, using one of SAS or R to access data produced in the other system is relatively straightforward and, in the case of using a DBMS, seamless. For any other data types, however, care must be taken and post processing is most likely necessary.

Handling the execution of R code from SAS using a pipe we have not encountered any difficulties. This has allowed us to integrate the R transcript in the SAS log during execution and writing the return code of the R function to the transcript and parsing it in SAS was easy. For the user of the SAS macro there is little indication that the processing is being done in R.

Finally, the decision to implement in only one language has already proven its value: we only needed to develop the functionality once and the SAS macros developed to transfer data can be re-used in other applications. In addition, we have already made one minor change to behaviour which required no changes to the SAS code and therefore the aim of reducing both development and validation effort has been achieved.

In conclusion, we have shown that with relatively small effort we can integrate R applications in SAS and expect that the opposite should prove to be as simple.

References

[1] Martin Gregory. Controlling opencdisc validator using r. *Pharmaceutical Programming*, 5(1-2):5–9, 2012-12-01T00:00:00.

[2] Inc. Artifex Software. Ghostscript 9.19, an interpreter for the postscript language and for pdf. http://www.ghostscript.com/, 2016. Retrieved 2016-08-07.

[3] PDF Labs. Pdftk server 2.02. https://www.pdflabs.com/tools/pdftk-server/, 2013. Retrieved 2016-08-07.

[4] SAS Institute. *SAS/ACCESS ®9.3 for Relational Databases: Reference, Second Edition.* SAS Institute, 2012.

[5] R Core Team. R Installation and Administration. https://cran.r-project.org/doc/manuals/r-release/R-admin.html, 2016. Retrieved 2016-09-26.

Acknowledgements

Contact Information

Martin Gregory
Merck
Frankfurter Str 250 64293 Darmstadt Germany
martin.gregory@merckgroup.com