And All With the Push of a Button !

Ray Pass, ASG, Inc.

THE INTRODUCTION

Picture this. You've got a ton of data sitting in SAS® data files located on a UNIX box at a different site in a different country, in fact on a different continent, but which can easily be reached via TCP/IP or some other network protocol. You've also got a set of supplementary control data sitting in a series of Excel worksheets on the network to which your PC is connected. The task is to create a system which will combine the SAS data with the Excel data, throw in a couple of manually-entered overall selection control parameters which will govern the ultimate mix of data, create a series of Microsoft Word reports from a pre-designed format (margins, fonts, etc), archive the reports on the network with appropriately updated report names, and have it all executed from start to finish by a group of professionals who cannot spell SAS and who can barely get Windows 95 going. Where to start? Where to go from there? Where to end?

This was the challenge. The answers were all in the SAS System, and were actually not that difficult to come by. The final product does it all, starting from a Windows 95 desktop icon which launches SAS with a start macro. The macro presents a simple data entry screen from which the user chooses some selection criteria. After this is accomplished, the macro brings in the Excel data via SAS/ACCESS® Software for PC File Formats. It then signs the user on to the remote UNIX box via SAS/CONNECT (after first checking to see if the user is already signed on – no sense doing it twice and getting an annoying little message that says, "You're already signed on, dummy!"). The Excel data, which by now exist as SAS data sets, are PROC UPLOADed to the remote platform, a ton of SAS processing happens there, and the final report image (PROC PRINTTO) is PROC DOWNLOADed back to the PC environment as a text file. SAS then opens Microsoft Word with the downloaded file as input and a Word auto-start macro which sets new margins, changes the font to a monospace one, repaginates the whole thing and saves it as a Word document with a properly updated name derived from the name of the input file. After printing the report, the user closes Word and the macro asks if another report is desired. If so, the whole thing starts again; if not, the user is signed off from the UNIX box, SAS closes and the user is returned to the Windows 95 desktop. The only encounters the user has with SAS are the screens asking for some selection input (even these are validated and error-checked) and whether or not the user is finished.

That's it! And all with push of a button! What else is there to say? Oh yes – the details. Well since this audience can spell SAS, we'll go into them a bit. This will still however remain at a global level. The details of the actual reports that are produced are not at all important because any output can be used. Likewise, the details of the input data are unimportant. I'll gloss over them a bit and then get into just how it all works, but most of the following will still remain generic. This paper is not meant to be of a 'tutorial' or 'lesson' nature, but rather of a 'show and tell' flavor. Sit back and take it all in, and then go out and run circles around my meager efforts. The tools are all fairly simple SAS techniques. When used in the right combinations however, they demonstrate some of the power available throughout the SAS System. It's that old Gestalt thing again. It works!

THE DATA

There were three main sources of data for this project as follows:

REPORT DATA: The real data that made up the heart of the final reports were all stored as SAS data sets on a UNIX box on a server in Europe connected to the local machine via TCP/IP. They were differentiated by country and month (part of the name of each data set), and were maintained by a separate functional group of people than the group preparing the reports. The reporting system described in this paper was only one of many that utilized this collection of data as input. The data existed in detail as well as summary versions and were uniform in nature as to data structures (variables, names, formats, etc.). The summary data sets ranged in size from tens of observations to hundreds, and the detail files ranged from hundreds of observations to millions. The summary files happened to be used in this application, although that is irrelevant other than efficiency (space, memory, etc) considerations.

These data were all processed by everyday SAS procedures (DATA steps, PROC MEANS, PROC SORT, PROC WHATEVER) in the main SAS programming portion of this system.

- CONTROL DATA: These data were all maintained in Excel worksheets on the local network to which the PCs were all connected. These data consisted of certain constants (currency exchange rates for example) and other country specific parameters. They were kept in Excel for ease of maintenance. Most business analysts are much more comfortable in a dedicated PC spreadsheet environment. Keeping these data here instead of in SAS datasets which could only be accessed (updated) via SAS procedures added to the friendliness and familiarity (read that as the separation from SAS) of the whole system. The SAS System provides many methods of directly accessing data from Excel worksheet files as well as many other PC file formats. In this application, the Excel files were directly read by SAS/ACCESS Software for PC File Formats. There was no necessity to have Excel up and running as part of the process (although that could have easily been built in by using a SAS tool other than SAS/ACCESS.) When using the SAS/ACCESS tool, data is data.

- SELECTION CRITERIA DATA: The last type of data needed to make the system work, and in fact to get it going in the first place, was a set of selection criteria which were chosen via dialog windows presented on the local machine (SAS Data step generated windows) displayed in the initial part of the process. In this case, a country and month had to be selected for each report. The windows were created with free input fields, although entries would only be accepted if they came from a defined list of possibilities. A screen listing the chosen criteria was then presented for final confirmation before proceeding.

THE FINAL REPORTS

The final reports had to be uniform and of a very specific format. Creating the report images was no problem for SAS – it was done via a DATA _NULL_ step with a PROC PRINTTO in effect after a significant amount of SAS procedure pre-processing. The reports were all to be archived as Microsoft Word documents (.doc format) in a specified subdirectory on the local network with file names which included the selection criteria (standard country abbreviations and month/year.) The end-users of the system were all familiar with Word, and storing the reports in this format made future retrieval and reproduction if necessary easy and painless. There are many routes that could be taken via SAS to provide final formatting characteristics such as including printer control codes in the body of the report, but the approach taken here was to simply create a Word auto-start macro which overlaid a specific list of formatting attributes to the input file after it was brought into Word. The archiving in the .doc format was also part of the Word auto-start macro.

THE CONSTRAINTS

The main constraint was that the real workhorse, SAS, was to be invisible to the end-user of the system. It was fine to use SAS to do all the work behind the scenes, but there was to be no real end-user interaction with the SAS System in creating the reports. The only interfaces between SAS and the end-user incorporated into the system were the selection criteria screens. Another constraint was that SAS/AF (SCL) was not to be used. It was desired to have the system written in simpler SAS techniques for future maintenance because SAS/AF was not being used as a standard development tool in this part of the organization. No problem. SAS always provides choices, so other techniques were used. Finally, the system had to work effortlessly and flawlessly, every time, with minimal possibility of error. It had to be reliable. It was, and still is.

THE FEATURES

The main required features of the system have already been mentioned above, but I'll repeat them once more here before going into the actual functional steps:

- The system had to launch from a Windows 95 desktop icon.
- There had to be minimal interaction, if any, with the real processing workhorse behind the scenes, SAS.
- The core data for the reports had to be easily accessed even though they existed on a different machine under a different operating platform in a different country on a different continent.
- The control data that was to be maintained by the end-users had to be kept in Microsoft Excel worksheets on the local network.
- There was a need for on-the-fly entry of selection criteria with error-checking and validation provided at that point.
- There had to be options for serial creation of multiple reports with different selection criteria, all in the same session.
- The reports had to be finally created and archived as Microsoft Word documents in a systematized storage

schema including automatically generated file names containing identification criteria.

- The system had to cleanly terminate returning the user to the Windows 95 desktop with all programs closed.

- The system had to be reliable. It had to work !

All the above features were accomplished using the SAS System. In fact, the SAS tools used are not particularly sophisticated. No SCL, no EIS, no FRAMEs, no real fancy stuff. That is not of course to say that the system could not have been created in a much more enhanced fashion with the SAS tools available, but we'll leave that as an open exercise for the audience.

THE SYSTEM (in semi-detail)

Now that the groundwork has been prepared, let's get to the real issue - how to do it. I'll lay out the steps in a sequential manner as seen from the outside. I won't provide the whole program by any means, but rather a collection of specific pieces of code, generic concepts, overall notions, etc. The entire process is presented in the diagram at the end of this paper (solid lines represent program flow; dashed lines are data flow.) Here we go.

The system starts with a Windows 95 desktop shortcut icon which opens the SAS System (version 6.12 was used) with an `-initstmt` invocation option as follows:

```
c:\sas\sas.exe -initstmt '%startmac;'
```

The `-initstmt` statement instructs the SAS System to execute the statement(s) contained in it's argument, after any found in any autoexec.sas file used. In this case, the system starts by executing a SAS macro called **`%startmac.`** In order for this macro to automatically start, a few steps must have been previously taken. Basically, SAS has to know about the existence of the macro you want to run automatically. There are a few ways to do this. In this instance, we used an autoexec.sas file which contained the following statements, among others:

```
filename maclib 'c:\maclib';
options  sasautos = maclib;
```

The **`sasautos`** option told SAS that we were storing user-defined macros in a file referred to by the fileref **`maclib`** (the actual path was c:\maclib.) In order to use the **`sasautos`** option, and therefore the SAS autocall facility, the SAS system option **`mautosource`** must also be set (this is a default setting, but may be manipulated as part of a configuration file, but we digress.)

So, at this point, we have a Windows 95 desktop shortcut icon set up which, when clicked on, will automatically start the SAS System with a user defined macro. To make it even more customized, we could have designed and used our own graphic for the icon instead of the one supplied by the SAS Institute, but we stopped at giving the desktop icon an appropriate name (call it "Anything You Like".)

The first thing that `%startmac` does is to define another macro called %**`checkit1`**. This is a macro that was given to me by the SAS institute, and looks like this:

```
%global signonrc;
*--------------------------------;
 macro to check if a signon link
 exists to a remote server.;
*--------------------------------;
 %macro checkit1;
    %let signonrc=0;
    %let rsub=%nrstr
               (rsubmit;
                %sysrput signonrc=1;
                endrsubmit;
               );
    &rsub;
 %mend checkit1;
 *--------------------------------;
```

It's used later in the process, but let's explain it now. Eventually we will want to establish a connection with SAS running on the remote server via SAS/CONNECT. When `%checkit1` is called, it checks to see if a connection has already been established. If not, then one is initiated at that point. This comes into play if the user chooses to produce more than one report in a work session. Only one signon operation, and ultimately only one signoff operation is necessary. The macro works by creating a macro variable called **`&signonrc`** with an initial value of 0. An attempt is then made to submit a command remotely (via the **`rsubmit`** statement.) If the attempt is successful, meaning that a connection does exist to the remote machine, the value of **`&signonrc`** is changed to 1; if not, it remains as 0; Later on in the program the value of **`&signonrc`** is checked. If it is 0, a `signon` command is given to establish the connection; if `&signonrc` = 1; the connection already exists and the `signon` command is not executed.

Now let's get back to our original macro. The first thing that `%startmac` does is to create a few selection criteria data-collection windows via a `data _null_` step with `window` and `display` statements. The criteria are saved as SAS macro variables. Using the window-display methodology doesn't provide nearly the power that SCL does, but with proper DATA step coding techniques, you can achieve error-checking, validation, macro environment interaction, screen customization, etc. In this case, the opening screen looked something like the following (this is a MUCH simplified version of the actual screen:)

```
Enter the country and month below from the
following list:

   aus  - Australia          jpn - Japan
   gbr  - Great Britain      swe - Sweden

   COUNTRY: ___
   MONTH   : _____   (MMMYY format)
```

Error trapping techniques were used via DATA step coding so as to not allow the process to proceed if country entries were misspelled (not from the prescribed list of allowable country abbreviations) or if month/year entries were invalid. A final confirmation box was presented which looked something like this:

```
You have entered: -->gbr<--  and  -->mar97<--.

If these are correct, enter Y and press ENTER.
If not, enter N and press ENTER to re-enter
data.

Enter Y or N here --> __.
```

An entry of 'Y' would allow the process to continue; an entry of 'N' would bring back the previous screen.

At this point in the process, the manually entered selection criteria exist as local SAS macro variables (`&country` and `&mmmyy`) and the next thing to do was to collect the control data which lived in a set of Excel files on the local network. SAS/ACCESS Software For PC File Formats was used for this operation. Maintenance of the actual data in the worksheets was completely independent of the report creation process. Certain step were taken to protect the integrity of the worksheet data. Non-data cells were cell-protected, and in fact, a shadow worksheet system was used such that the actual worksheets that were brought into SAS were protected (uneditable) copies of the worksheets that were actually manually maintained. But that is a topic for an Excel paper, not this forum.

The result of a PROC ACCESS operation is a SAS data view. After one view was created for each worksheet file, SAS data files were created, one per view, so that some additional processing could take place on the data values. This had to do with formatting, minor data conversion, etc.

At that point in the process, the `%checkit` macro was called. If no connection previously existed to the UNIX box via SAS/CONNECT, the program issued a `signon` statement and the connection was established. This was always the case for the first, if not the only, report to be created in a session. The `signon` process produces data entry boxes calling for the user's login ID and password. In order for the connection to be established, some prior steps had to be taken. This work was actually done via the autoexec.sas processing and was therefore available for this system. The necessary statements which were included in the autoexec.sas file were specific versions of the following generic statements;

```
%let server1 = 1.222.33.4;
filename rlink
'c:\sas\connect\saslink\tcpunix.scr';
options  comamid=tcp remote=server1;
```

The purpose of these statements is to set up a default logon script for the remote server that you wish to connect to. The details are not important for the overall flow of this paper. Suffice it to say that at this point in the process, a connection was established to the remote server and remote compute services were available.

The next step was to transfer the SAS macro variables `&country` and `&mmmyy` from the local SAS environment to the remote SAS environment where they would be needed for the bulk of the report processing. SAS provided us with a utility macro for just this purpose, `%syslput`. You can find documentation for this macro in the new <u>SAS Macro Language: Reference</u>, but it won't actually become part of the SAS System provided autocall macro library until Version 7. The edited version of the macro code that was used in this application is as follows:

```
%-------------------------------;
  %macro syslput(mv,val,remote=);
    %global mvar value thost;

    %let mvar  = &mv;
    %let value = &val;
    %let thost = &remote;

    %let go = %str
           (rsubmit &thost;
              data _null_;
                call symput("&mvar",
                            "&value");
              run;
            endrsubmit;);
    &go;
  %mend syslput;
%-------------------------------;
```

`%syslput` works by assigning the value of a local macro variable (stored in the local environment) to a macro variable stored in the remote environment. If you are only connected to one remote server, you don't have to identify it; other than that, the syntax is pretty straightforward. The two statements used here were:

```
  %syslput(country,&country);
  %syslput(mmmyy,  &mmmyy);
```

The first argument in each statement, the one without the ampersand, is the name of the remote macro variable you are writing into; the second one is the local macro variable that contains the value you wish to assign to the remote macro variable. The macro submits a **call symput** statement (housed in a **data _null_** step)_ to the remote server. This in turn creates a macro variable on the remote machine. The mechanism works by using local macro variable values as the arguments to the remote **call symput** statement. There is a corresponding SAS macro statement to go the other way, namely `%sysrput`. We didn't need it in this process, but as you might expect, it's purpose is to transfer values from remote SAS macro variables to local variables. Logical, isn't it.

At this point we were ready to suspend processing on the local machine and start processing on the remote machine via SAS/CONNECT and remote compute services. We initiated this with the **rsubmit** statement. This told the system that the following statements encountered, until the appearance of the **endrsubmit** statement, were to be processed on the remote system. The first processes to happen on the remote box were the uploading of the local SAS data sets created from the Excel data. This was accomplished via the SAS/CONNECT **upload** procedure. Once these data sets were uploaded to the remote box, another macro which existed on that platform (let's call it **%bigmac)** was called to actually create the report. As part of that process, which actually consisted of quite a bit of data manipulation culminating in a large **data _null_** step, a **proc printto** was put into effect to produce a final text image of the final SAS created report. Let's call this file, in UNIX file syntax, `"~/unix/&country&mmmyy..lst"` (the `&country` and `&mmmyy` macro variables resolve to the appropriate country and month in the actual data set name.) This file was then downloaded via the SAS/CONNECT **proc download** statement to a text file on the local PC network (let's call this file, in PC file syntax, `"c:\reports\&country&mmmyy..lst"`. As the PC files were sometimes to be updated with new data, no check was done to see if the PC file already existed. If it did, it was simply overwritten. Processing was now complete on the remote box and so the **endrsubmit** command was issued. These last few steps can be represented as:

```
 rsubmit;
    proc upload inlib   = work
                outlib = work;
         select excel1 excel2 excel3;
    run;

    %bigmac;

    proc download
         infile =
         "~/unix/&country&mmmyy..lst"
         outfile =
     "c:\reports\&country&mmmyy..lst"
    run;
 endrsubmit;
```

Note that SAS data sets were uploaded via the **proc upload** statement and that text files were downloaded via **proc download**. Both types of files, as well as SAS catalogs, can be sent either way.

At this point, remote processing on the UNIX box was complete and control was returned to the PC. In keeping with the automated nature of the system, SAS then opened up Microsoft Word. A few SAS options were previosly set, namely **noxsync** and **noxwait**. These two options have the combined effect of separating the operation of SAS from the operation of the PC program launched. SAS does not have to wait until the PC program finishes before it resumes its operations (**noxsync**) and it is not necessary for the user to type EXIT to fully close the PC program and return control to SAS (**noxwait**). These options are not always necessary, depending on the particular situation, but were found to enhance the smooth and automatic nature of the process in this system. The SAS **x** statement was used to launch Word as follows:

```
x "c:\reports\msword.bat
  &country&mmmyy..lst  /mstart";
```

It was found that the smoothest operation of the system necessitated that Word be started via a PC batch file (**msword.bat** in this case) which contained two arguments, **%1** and **%2**. The batch file therefore contained one line, as follows:

```
C:\WORD\WINWORD.EXE %1 %2
```

The first argument, **%1**, contained the name of the PC file to be used as input to the Word program (opened automatically upon startup of Word), in this case **&country&mmmyy..lst**, with the substitutions of the chosen country (**&country**) and month (**&mmmyy**). The second argument, **%2**, contained an instruction which told Word to apply the Word macro called **start** (any name can be used) upon initiation of the program. This Word macro can contain any Word instructions and must be installed on the PC previously to the operation of the system. This process is really quite easy however – it basically amounts to copying a file, with a few small added details. In this case, the Word auto-start macro **start** contained instructions to set a display font (monospace Courier was used to assure correct display of tabular data), set desired margins, and most importantly, to save the file brought in to Word as a Word document. If the text file being brought in resolved to **gbrmar97.lst** for example, a Word formatted **gbrmar97.doc** file was saved to the appropriate subdirectory on the PC network. With the PC now displaying the Word program and the desired report on the screen, the user had the option of printing at that point or at some later point (the file was now saved as a Word document.)

The next step was for the user to close Word. Control then returned to SAS which then displayed another window created with a SAS **data _null_** step containing additional **window** and **display** statements. The window looked something like this:

```
Do you want to run another report?

If you do, enter Y below and press ENTER.
If not, enter N below and press ENTER.

Enter Y or N here ---> _
```

An entry of **Y** would cause the program to return to the first data entry screen via a label at that early part of the program. This was easily accomplished by making the response to the above screen a macro variable and using **%if-%then-%goto** code. An entry of **N** would alternately cause two other SAS commands to be executed. First a **signoff** statement would be submitted to close the connection to the remote server (good housekeeping), and then an **endsas** statement would be submitted, closing SAS neatly and returning the user to the Windows 95 desktop.

That's it. A lot of processing going on with not much effort on the part of the end-user.

THE SYSTEM (in pleasant summary)

Now that we've seen the system in somewhat of a detailed manner, here it is in a sequential summary fashion from beginning to end:

1. User clicks on a Windows 95 desktop shortcut icon which opens SAS with an initial statement. This statement tells SAS to begin executing a stored user-defined macro called **%startmac** (any name you like.)

2. **%startmac** displays a selection criteria data-entry screen to collect desired country and month for report.

3. Entered data is error-checked for valid entries.

4. Confirmation screen is presented to confirm selection criteria data entered. Steps 2-4 are repeated if necessary until data entered is confirmed.

5. Selection criteria data are translated into local SAS macro variables.

6. Control data are collected from Excel worksheet files via SAS/ACCESS Software for PC File Formats. These data are represented as SAS data views.

7. Data sets are created from the data views with further manipulation of the collected data.

8. Check is made to see if a connection exists between the local SAS-PC session and SAS on the remote UNIX box.

9. If no connection exists, one is established using SAS/CONNECT with previously entered connect parameters (via autoexec.sas file.)

10. Local SAS macro selection criteria variables are transferred to the remote environment via `%syslput` macro.

11. Remote computing is initiated via `rsubmit` statement.

12. Data sets created from the Excel data are uploaded to the remote box via `proc upload`.

13. Main report creation program, contained in macro `%bigmac` on the remote box, is executed. This process culminates in the creation of a text image of the final report, via `proc printto`.

14. Text image is transferred down to the PC via `proc download`.

15. Remote processing is terminated via `endrsubmit` statement.

16. Microsoft Word is automatically launched from SAS via the SAS `x` statement. Included in this statement, via a PC batch file, are instructions to automatically open the now downloaded text image of the finished report. The batch file also includes instructions for Word to use an auto-start macro which reformats the opened file according to stored specifications and saves the report as a properly named Word `doc` document.

17. User prints the report at this time if desired, or reserves printing the now saved Word document for a later time.

18. User closes Word.

19. Screen is presented asking if creation of another report is desired at this time. If this is desired, the process begins anew with the selection criteria data-entry screens. If not, the remote connection is terminated, SAS closes and the user is returned to the Windows 95 desktop.

20. Finis.

THE CONCLUSION

Well, that's it. Much was accomplished by the unsuspecting end-user without even knowing what was actually happening. By using the power inherent in the SAS System, a large amount of data manipulation and report creation was done with disparate data types on independent sources. From the end-user's perspective however, the whole process was indeed "friendly" because the interfaces "apparently" used were those that were known and trusted old friends. SAS provided the means of keeping the process removed from the user, while performing all the work "invisibly." Valuable, productive, updated reports were produced and stored, easily and most importantly, all with the push of a button!

THE ACKNOWLEDGEMENTS

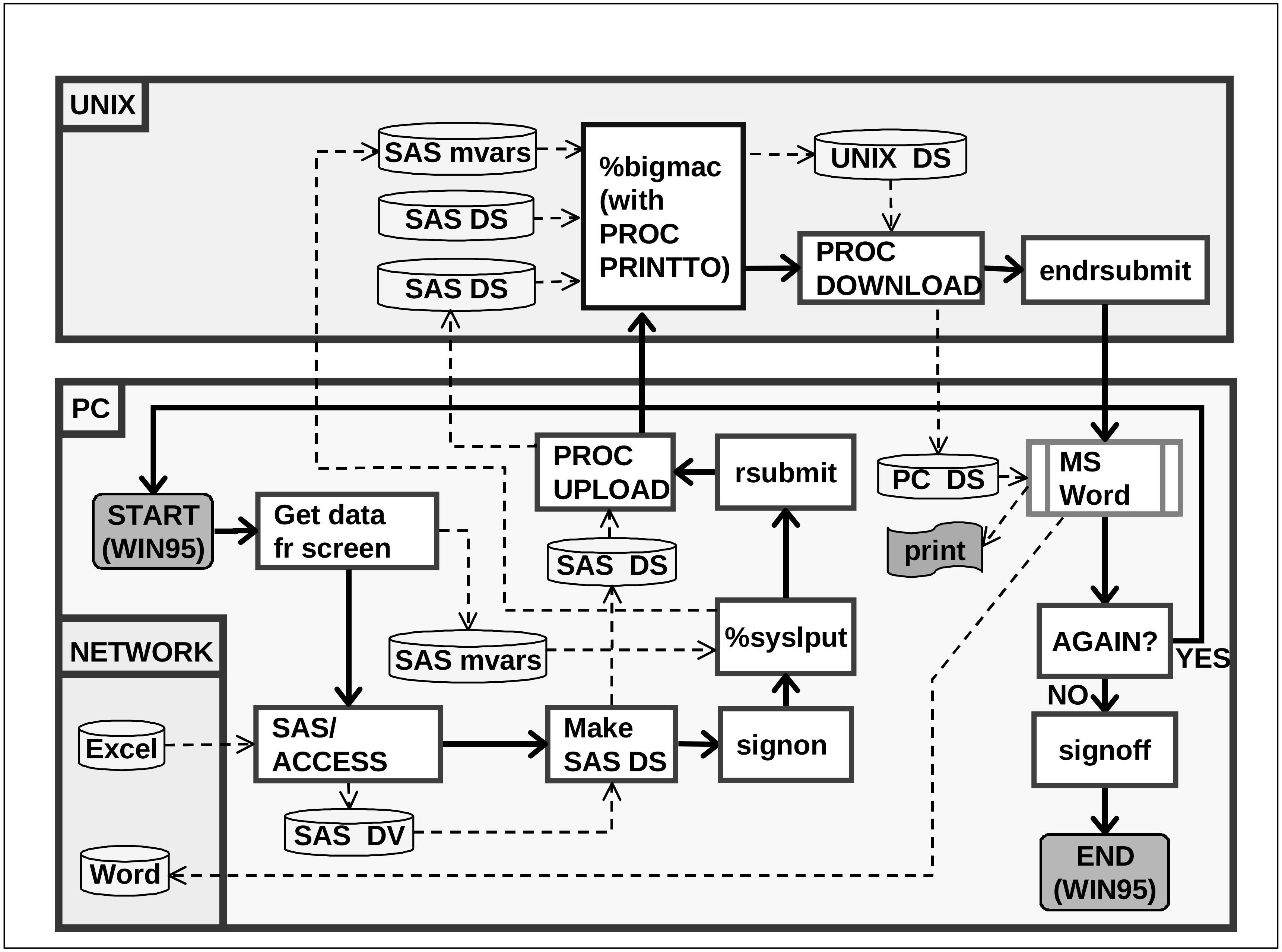
I would like to thank Don Wdowski and Greg Barnes Nelson for technically reviewing this paper for me. If there are still minor mistakes, it's because I made more changes after they saw the "final" version.

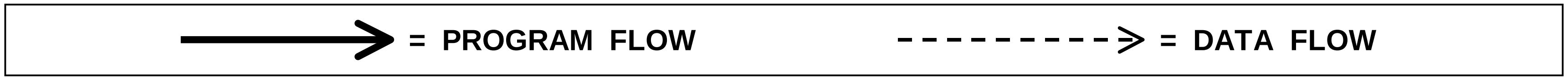
THE AUTHOR

Ray Pass is a Senior Consultant at:

ASG, Inc — (203) 356-9540 (voice)
1100 Summer Street — (203) 967-8644 (fax)
Stamford, CT 06905 — raypass@worldnet.att.net

And All With The Push of a Button !


UNIX
SAS mvars
SAS DS
SAS DS
%bigmac (with PROC PRINTTO)
UNIX DS
PROC DOWNLOAD
endrsubmit
PC
PROC UPLOAD
rsubmit
PC DS
MS Word
START (WIN95)
Get data fr screen
print
SAS DS
SAS mvars
%syslput
AGAIN?
YES
NO
NETWORK
Excel
SAS/ ACCESS
Make SAS DS
signon
signoff
SAS DV
Word
END (WIN95)


= PROGRAM FLOW
= DATA FLOW