"YOU WANT THIS NEW APPLICATION TO RUN ON OUR VAX, PC AND SUN COMPUTERS???!!!"

(OR CROSS-PLATFORM APPLICATION DEVELOPMENT USING SAS®)

David Franklin, Mainridge Management Ltd, London, UK

ABSTRACT

The need for cross-platform applications is increasing as individuals and companies are becoming aware that the future of their own in-house or packaged applications need to be able to work on two or more operating systems, or migrate from one operating system to another quickly and with ease. Investment in time and resources to develop a new application on an existing operating system is often lost when a new operating system is brought in. Rather than loose that investment many adopt the attitude of keep the old operating system going while new software is to be developed on the new operating system. Others though spend significant amounts of time and resources often rewriting from scratch new applications doing the same job that the old application did but in the new operating system.

SAS®, though its software and Multi-Vendor Support environment, addresses these problems and brings together the opportunity of creating a single SAS/AF® application on one operating system and distributing that single master to any one of the numerous other operating systems that the SAS® System supports, and have it running in a matter of minutes, and not days, or even months as occasionally occurs in other development environments.

INTRODUCTION

So you have just had a meeting with your manager and she wants you to build a data warehouse and mining system. The databases are in Boston and Los Angles, USA; Glasgow and Edinburgh, Scotland; Nice, France; and Singapore. A usual enough request, but for one thing - she wants the new application to run on the Digital Alpha and VAX operating systems in the UK, the Sun UNIX operating systems in the US, and local Windows on network and standalone PCs. There were also one or two more things that were said as you were walking out the door in a daze, "That system has to last ten years; I don't want to spend any money on development when we change operating systems in the applications' lifetime; I don't want it crashing at 00:01 on the 1st January 2000; and those people in the UK want to use that odd-ball operating system of theirs in 18 months time". Quite a tall order for any solution, whether it be SAS®, C, Java, Visual Basic, or any other of the myriad of development languages.

For the purposes of this paper I will concentrate on a SAS® solution and give a brief introduction to some of the challenges that arise.

WHY SAS®?

Why even look at SAS® as a solution to this challenge?

Due to the initial requirements, our goal is to write not an application for every operating system, but a single application that will run on ALL the operating systems asked for. With only minor code additions and proper planning, it is possible for the initial application on its host operating system to be up and running on a different operating system that SAS® supports now and in the future, within minutes.

There is another factor that makes SAS® a good option for business use - 00:01 on the 1st January 2000, or more commonly known as the millennium bug. As SAS® stores the full date as the number of days from a base of 1st January 1960, the 1st January 2000 will be stored as just another day from that base date. Just as a note though, SAS® will have a problem with dates on the current SAS® version releases, around the middle of the seventh millennium, about 5500 years from now. As a result there is no problem with SAS® dealing with the year 2000.

TWO OPERATING SYSTEM VARIABLES AND A FUNCTION

When designing a single application that will work on different platforms planning is critical. Planning will either make a new application successful or lead it to disaster.

The analogy of a designer designing an "off the shelf" kit set house is a good one. The dimensions will be the same, the windows will be the same, they will be painted the same but there is always that little piece of the kit that makes each house look alike but just has that little something different, whether it be a shower instead of a bath, or the latest dishwasher instead of the standard model.

This is the same way a Developer designs an application which is to go on two or more operating systems - the tasks that will be performed and the look and feel will be the same and it is only the operating system features that

will be different. With SAS® though, the list of operating system features is few and usually restricted to operating system file and print management facilities. The rest of the application code will be operating system independent. It is crucial that in order to build a single application which will work on two or more operating systems, the developer identify what is dependant and independent modules and place them correctly within the application so that the dependant modules can be easily edited to accept new operating system dependant code without having to edit modules that contain independent code.

Fortunately with SAS® there can exist two or more operating system dependant code modules within the one application. The operating system in action at the time will determine automatically which module should run. The value of the SYSSCP automatic macro variable displays an abbreviation for the operating system in use and it is this that will determine which module should run. Below is a fragment of a program creating a directory whose name is stored in the local variable _DRCTRY under a Windows or VAX operating system.

```
if &sysscp eq 'WIN' then
   _cmdtext='md '||_drctry;
else if &sysscp eq 'VAX' then
   _cmdtext='create/dir
[.'||trim(left(_dirctry))||']';
system(_cmdtext);
```

Example 1: Use of the SYSSCP macro variable to run operating system dependant code

Another common question that is asked during planning is what versions of SAS® shall the application support. This issue is becoming more important today, especially between the older versions of SAS® that only support a non-GUI environment and newer versions of SAS® that support only GUI environments. It is not the GUI/non-GUI displays that are important, but more the new SAS® language functionality that is becoming available in the later versions that makes the code faster and more compact than was available in earlier versions. In an application there is a need to both write the code for the earlier versions of SAS®, while still using the developments that are available in the later versions. Within SAS® the automatic macro variable SYSVER displays the version of SAS® that the current session is running in. Below is a fragment of a program reporting a list of pupils with an Exam Date and Grade, and also what university they have said they want to go to.

```
if &sysver ge 6.11 then do;
   submit continue;
      proc report data=work.examdata nowindows;
         column pupil examdate grade varsty;
         define pupil /group 'Pupil' id;
         define examdate /order order=internal
            'Examination Date' format=date9.;
         define grade /display
           'Examination Grade';
         define varsty /display
           'University to attend';
      run;
   endsubmit;
end;
else if &sysver le 6.10 then do;
   submit continue;
      proc sort data=work.examdata;
         by pupil examdate;
      run;
      data work.reptdat1;
         set work.examdata;
         if not first.pupil then pupil='';
      run;
      proc print data=work.reptdat1 label
                     noobs;
         var examdate grade;
         id pupil;
         label pupil='Pupil'
               examdate='Examination Date'
               grade='Examination Grade'
               varsty='University to attend';
         format examdate date9.;
      run;
   endsubmit;
end;
```

Example 2: Use of the SYSVER macro variable to run version dependant code

With the above example, the report produced will be the same in both cases, it is only the fact that the later version (6.11) now supports the ID option in the DEFINE statement of the REPORT procedure that makes the code different and faster to run, which was not available in earlier versions.

Also under consideration when planning is the SAS® modules that will be available. This is an important issue since what the application is asked to do will depend on the SAS® modules available to it. If a module is not present then there are two possible outcomes - either the task can be carried out another way or cannot be done at all. There is a macro and SCL function that allows the developer to determine whether a product is present or not, called SYSPROD.

STARTING THE CODING PROCESS WITH SAS®

Planning of an application is always a decision based upon personal preferences. Often though there is too much "planning" documentation, and occasionally too little, with both achieving the same result - disaster. There is fortunately a happy medium but this depends on a number of factors, including:

- the size of the project concerned.
- the amount of the documentation that will be required to build AND maintain the new application.

This last factor is the most important since it is this that will determine the applications future. Too little and the application may not do what it was designed for, and if it gets that far, maintaining it would be difficult if not impossible. Too much and both the builder and maintenance will have to wade through mountains of paper, often with contradictions, causing delays and/or an application which is not quite what the users wanted.

However any design of an application will have three common items:

- what the user would like to see (output and usage).
- where the data is coming from.
- a very basic application flow.

Now that the basic design documentation is done, now it is time for creating the SAS/AF® application. Remember that there is only one application that is going to be written, but will be distributed among several operating systems so there are several considerations that have to be thought of before actual coding can begin. Basic considerations are:

- is the application going to be on GUI or non-GUI systems, or both, and on what operating systems.
- the SAS® versions that will be supported.
- the SAS® modules that will be available.

While the considerations of operating systems, SAS® versions and modules to use can be overcome with the methods described earlier, the question of whether the application is going to be on GUI or non-GUI systems, or both, is still a challenge. An important consideration since it will affect such things as whether you can use colours, mouse support, and even screen size. If screen size is different among the operating systems you should write the application for the smallest size available. Also affected is the type of SAS/AF® entries which you can use. If it is an all GUI approach then all SAS/AF® entries, including FRAME, PROGRAM, SCL and CBT entries can be used. If however there is the support for GUI and non-GUI operating systems, or all non-GUI operating systems, then FRAME entries can not be used as these are not supported in a non-GUI environment. However, there is little lost if you cannot include FRAME entries in the final application as SAS® itself will use internal techniques that make PROGRAM entry icons, buttons etc. appear in a GUI format wherever the operating system will allow it.

Another decision that must be made when the application is ready to build is the directory structure that will contain it. This is critical as the location of the application system and data files, along with any others, are needed. Typically an application would have a structure of:

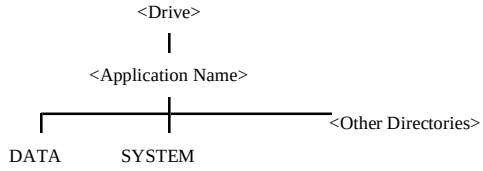


Figure 1: A Common Directory Structure for an Application

The application itself is built inside a SAS® Catalog with at least one external program file and possibly a series of SAS® datasets. The following figure shows a basic structure that is common when building a multi-platform SAS/AF® application:

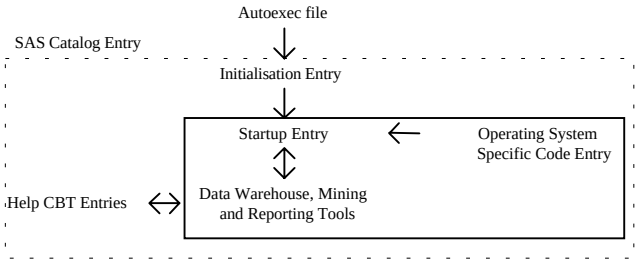


Figure 2: A Basic Application Structure

Clearly during start-up of the application two things should happen. These are:

- run start-up
- initialise the system if no start-up or application runtime files present

First looking at starting up an application, one method is to have an AUTOEXEC file give the location of the SYSTEM directory under the Application Folder and issue a SAS/AF® DM command to begin a start-up file in the application catalog. This later file will contain additional libname definitions as well as access user options. These may be default LINESIZE and PAGESIZE options, just to name a few. If the files containing such information are not found then the

system should initialise these files itself bringing in any default information as necessary.

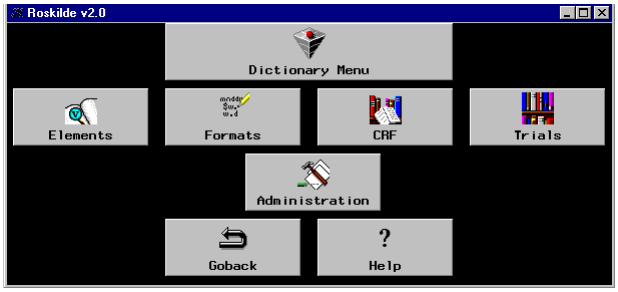
But the key challenge here is how exactly is the operating system dependant code going to placed around the operating system independent code in such a way that there is going to be one catalog entry containing the former code? During the writing of the application it will become apparent what code is operating system dependant. This code would then be placed in a separate catalog SCL entry and linked by the use of the METHOD SCL statement. Within each METHOD block in the SCL entry there would exist a series of statements similar to that in Example 1. Thus when during runtime of an application it encounters code that is operating system dependant, the METHOD statement in its place would link to the METHOD block in the SCL entry relating to operating system dependant code and execute what was needed depending on the operating system running.

What if later, the application was being sent to, or made ready to be used on, another operating system? Because the operating system dependant code is in a single SAS® catalog entry and there is hopefully little of it, it would be a simple case of what is there and adding to it. There would be no reason to hunt through all the application entries looking for dependant code and changing it where necessary which is time consuming and prone to error. Looking back at example 1, if the code to create the directory for the VAX was not present then it would be a simple case of adding the additional code as shown.

While coding, there are two useful items that make building the user interface easier.

The first in the use of PMENUs. Within an application window, the developer can define a PMENU to use, as defined earlier in a PMENU procedure. This has the advantage of relying on SAS® to position the PMENU correctly within the user window, as well as relying on SAS® to decide the size of the buttons etc., therefore allowing system independence. In the past many user screens have had their PMENUs defined by placing actual buttons on the screen where a PMENU would normally go.

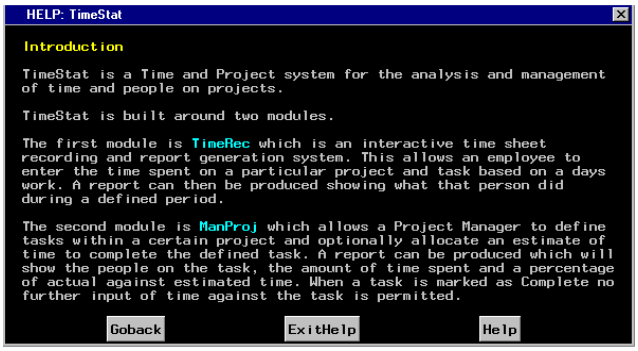
The second item is the use of the BLOCK statement. Many people build FRAME entries similar to Example 3 which take an excessive amount of time to build especially to get the icons the right size and place. The block statement, while a PROGRAM entry, will do the same job and have the benefits of using icons in a GUI environment and blocks in a non-GUI environment, a SAS® runtime decision, as well as doing the size and placement on the screen of the icons or blocks.



Example 3: Example of user view of BLOCK function output

ONLINE HELP SCREENS

The development of Online Help screens is often the most overlooked or neglected of all the phases of application development however it is increasingly the only help that a user will have to navigate through an application. SAS® itself has a very good facility of building help text through SAS® CBT catalog entries. An example of such an output text is in Example 4.



Example 4: Example of user view of Help Text

While the format of the output text is not up to that available on standard Windows help operating systems, it is still advanced enough to allow the developer to design and build hypertext links and be operating system independent at the same time, therefore there is no need to create a new set of help text based on operating system dependencies. Formatting attributes include colour and highlight attributes.

Developing an application which will work in different spoken languages is difficult, however almost all users will accept an application if the help text only is in their native language. To switch the language of the help system for a particular user is easily carried if you have as part of your distribution disk a set on SAS® CBT catalog entries in different catalogs (one for each language and the same entry name) and simply copying the entries for the desired language catalog into the application catalog during application initialisation. It is

also possible to have a menu option in a Set-up Window which when changed will do the same task after initialisation but this means placing these catalogs in the SYSTEM folder during loading the system onto a computer, however these catalogs are usually small and are not a space consideration issue.

SENDING THE APPLICATION TO ANOTHER OPERATING SYSTEM

Before you there exists the complete and tested application. It is now time create a distribution disk to be made available to other potential users of the application. The aim is to have two files only, plus an additional file with installation notes and comments regarding the release which are not documented elsewhere. These notes though should be minimal and only include notes specific to the target installation. This last file would normally be a text file.

The first of the two files that are to be sent is the default AUTOEXEC.SAS file which is the file that is run whenever the application is started. This would be a straight text file.

The second file present is the application catalog in a CPORT procedure transport format. All datasets etc. which are needed for the system to run, including datasets containing options data, would be created during the initialisation phase and any default values would be set during that time, therefore these would not be included in the transport format file.

This distribution disk would then be sent to the target operating system, the text files copied and the transport file processed using the CIMPORT procedure. Providing planning and coding of the application was done correctly and the system dependant code is correct the application itself would start immediately.

The scenario where you have an existing application on one operating system that you would like transferred to another operating system is similar.

Before creating the transport disk, the catalog entry containing the operating specific code would be amended to include the target operating system, if it didn't already. It would be then a case of copying the AUTOEXEC file from the old operating system to the transport disk making any changes where necessary. Then using the CPORT procedure on the SYSTEM and data directories, a transport file would be created and placed on the transfer disk. This transfer disk would then be sent to the target operating system, the text files copied and each of the transport files processed using the CIMPORT procedure. Providing planning and coding of the application was done correctly and the system dependant code is correct the application itself would start immediately.

CONCLUSION

Through proper planning and using the SAS® System, it is possible to develop applications which can run on several operating systems and versions from the one master source on a single operating system. With this achieved, it is possible for in-house or packaged applications to work on two or more operating systems, or migrate from one operating system to another, quickly and with ease. The small additional investment in time and resources to develop a new application on an existing operating system is saved many times over when that same application is migrated to a new operating system, often taking only hours instead of months or even years. It is for this reason that many individuals and companies are looking at SAS® for their application development environment of choice.

CONTACT

David Franklin
Mainridge Management Limited
2 London Wall Buildings
London Wall
London EC2M 5PP
United Kingdom
Ph: +44(0)171-628 4200 or +44(0)956-395197(mobile)
Fax: +44(0)171-588 2718
E-mail: 100316.3451@CompuServe.com