

Cross Platform Application Development in SAS

Exploitation of Multi-Vendor Architecture™ (MVA)

Gregory S. Barnes Nelson
ASG, Inc.

SAS is a registered trademark or trademark of SAS Institute Inc.
in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or
Trademarks of their respective companies.

Abstract

Developing applications in one environment and porting them to a second can be a tremendous effort. The advantages offer off-site development opportunities and independence among developers. In addition, it allows the development effort to be done in an environment comfortable to the developer. The pitfalls and extra effort required to ensure a successful project tend to scare most people off. This paper will highlight some of the issues in portability and cross-platform development and offer strategies for successful implementation of this model of development.

Introduction

In 1990, SAS Institute announced a new generation of software, which would break the barriers of cross-platform development and allow the SAS System to be virtually independent of the hardware on which any SAS application was developed (SAS Communications, 1st Qtr. 1990). Multi-Vendor Architecture™ (MVA) was born with Version 6 of the SAS System. A new design strategy enabled most (90%) of the SAS System to be independent of the hardware with the other 10% being host dependent features that would take advantage of operating system on which SAS would be deployed.

Almost a decade later, we can still appreciate the effort that SAS Institute put into the port of the SAS System from PL/I to the C programming language as we deploy our SAS programs and interactive applications on nearly two dozen platforms. Daily, we tout the strength of the SAS System as a true enterprise application. Hardware and operating system independence has allowed us to develop in one environment and distribute across most all modern systems. In the age of tele-commuting and offshore development practices, it is clearly one of the strengths that sets SAS apart from almost all development and data warehousing solutions.

As a real-world example, consider the fact that although developers at SAS Institute have developed most of the SAS System on HP-UX UNIX systems, many of these developers are now moving their development work to Microsoft Windows NT as their platform of choice.

The focus of this paper is to examine the claims made in the SAS Communications article in 1990. Specifically, this paper will discuss just how easy is it to take an application developed on one platform and port/ deploy it on another. We will discuss coding strategies to ensure application portability, host-specific features, data storage and access mechanisms, code and output storage mechanisms, design considerations, porting and data transfer issues, as well as general client/ server development issues. The goal of this paper is to provide a blueprint of the land mines to avoid when tackling a remote application development project or in consideration of a scalable SAS solution.

Coding Strategies

Concentration on coding techniques has probably been the single most popular sub-topic for SAS conference papers on the general topic of multi-vendor architecture and cross-platform portability of SAS applications. It is the meat and potatoes of the cross-platform development platter. It is what constitutes the majority of that 10% of the SAS System that jumps out at us when we move from one system to another.

Startup conventions

Whenever developing a new application, whether a batch analysis program or an interactive SAS/AF™ application, I try to segment most of the code for the operating system that it is being run under into the mechanism that launches the program(s). It is here where SAS meets the operating system. It may be easier to identify the files and directories that you use by taking advantage of the operating system. For example, in OpenVMS, you may define this with a command file; in UNIX, perhaps you'll use a shell script; in MVS you may use JCL to point to your physical files and libraries. However, I would suggest that instead of referencing the files in the operating system native code, use SAS to maintain the connections between the operating system and the SAS System.

Most often, we do this by putting the file and library references in an autoexec.sas file or a startup SCL program. This gives us one point of contact for our connection with the operating system and the subsequent file system. When we port the application to another environment, we don't have to worry about changing much beyond the file and library references found in the autoexec.sas or startup SCL file. The simplest way to do this is to just name the path (or at least the root path) in a macro variable, as in:

```
%let path="c:\mydata\clients\asg";
```

Now, the &path macro variable will resolve throughout the application to the correct starting location and all we have to worry about is maintaining that directory structure.

In Releases 6.09e and 6.11 of the SAS System (and later), we can take advantage of the %sysfunc macro function in conjunction with the getoption function to identify the location of the autoexec file, if one is used. For example, if we were to start SAS with the -autoexec invocation option, we could use specific code to identify the location of the autoexec.sas file. If this file was at the root of our application directory structure, we could use this information to form a macro

variable naming convention to identify our paths for libname and filename statements.

To launch SAS in UNIX:

```
sas -autoexec /project/myauto.sas
```

To launch SAS in Windows:

```
sas -autoexec c:\temp\myauto.sas
```

We can use the following code to capture the name of the autoexec file we ran when we launched SAS:

```
%let path=%sysfunc(getoption(autoexec));
%put %path;
```

In UNIX, the macro variable &path, would resolve as:

```
/project/myauto.sas
```

In Windows, the macro variable &path, would resolve as:

```
C:\temp\MYAUTO.SAS
```

This is a nice way to avoid having to code the actual path names into your SAS code and still get the same results. We could then parse the macro variable to strip out the *.sas part and end up with the root directory of the application.

Another way to get around having to hard-code the actual path names in our SAS program is to set a macro variable upon invocation, either in our config.sas file or via the invocation command. For example, our invocation might have the following parameters:

```
sas -set approot c:\myapp
```

Later in our code, we could reference libname and filename statements as in:

```
Libname maclib "&approot/maclib"; /* UNIX */
Libname maclib "&approot\maclib"; /* Win */
```

In SCL, we could reference a library with the following type of syntax:

```
/* store the value of the macro variable into
an SCL variable */

path=symgetc('approot');

/* allocate the Macro Lib as read-only */

sysrc=libname('maclib',path||"\macros",'',
"access=readonly");
```

Later, we'll explore how to take advantage of macro variables and SCL calls to figure out what system we're on and perform more complex macro resolution to identify paths for our application (and also to change the direction of the slash for Windows vs UNIX). It's important, however, to first note what we need to be concerned with in cross-platform development.

Libname

A SAS libname statement tells SAS where to find data that is to be used in the application. A library (as defined by that which is

referenced by a libname statement) can contain the following types of information:

- A single sequential data file like a proprietary data set (spss engine for example):

```
libname myfile spss 'c:\temp\spss.por';
```
- a non-native binary SAS dataset (sassfio engine):

```
libname windata sassfio '/winnt/sales.sd2';
```
- a SAS data library consisting of SAS datasets, views, catalogs, mddb, or fddb files

```
libname store 'b'; /* CMS */
libname store 'company.datalib'; /* MVS */
libname store 'c:\comp\datalib'; /* Win */
libname store '/company/datalib'; /* UNIX */
libname store 'store: '; /* VMS */
```

In future versions (Version 7), we will also see the addition of relational database systems (RDBMS) into this category where we will identify the database, such as Oracle by using the libname statement, and then just reference the table name as we would any two-level SAS data set name.

Because we anticipate the potential relocation of the libraries and put them into the autoexec.sas or startup SCL program, we can now easily modify this single file at any time.

Filename

The filename statement in SAS is an alias or pointer to a physical file contained on some file system. In SAS, we use filename statements to reference external files that we can either read from or write to. Like libname statements, they tell SAS something about the file system and therefore are important when moving from one platform to another. Also like libname statements, these should go into the autoexec.sas or startup SCL files.

Examples of filename statement on various platforms include:

```
Filename indata 'mydata data b'; / CMS*/
Filename indata 'EDC.MYORG.MYPDS'; /* MVS */
Filename indata 'c:\tmp\data.txt'; /* Win */
Filename indata '/asg/my_data'; /* UNIX */
Filename indata 'Data:mydata.xpt'; /* VMS */
```

Again, we can use macro variables to identify the path names instead of hard-coding the names throughout our programs.

Device

The device system option specifies an output device driver for SAS/GRAPH® software. Whether we want to display the graphics interactively, to a particular printer or to a specific file, we again have to be concerned about this system option. We put it in the autoexec.sas or startup SCL code to ensure that we won't have to deal with this throughout our application conversion. A nice trick here is to create a

applications development

macro (let's call it %startup) which contains all of our startup options. Below is an example of a startup macro that sets the defaults for a host of system or environment specific components. Here we only have to compile the macro once and store it. The execution of this macro handles all of the common activities.

```
%macro startup / store des='Sample startup
macro';

%global excel secure fin temp datlib oorpat
planclr oorclr;

options noxwait noxsync;

/* location of Excel */

%let excel="C:\Program Files\Microsoft
Office\EXCEL.EXE";

/* special security for users */

%let valid = 74, 1936; /* valid values */

/* graphics catalogs used */

%let temp = work.gseg; /* catalog for
temp components */

%let fin = appuser.grseg; /* cat for
graphs & templates */

%let oorpat = M4N135; /* pattern for
background /

%let oorclr = red; /* color settings for
background */

%let planclr = green; /* color settings for
plan */

/* data library for datasets and formats
*/

%let datlib = appsd1;

/* get system date */

%let date=%sysdate;

%mend;
```

Once compiled, we can simply call the macro in our startup routine instead of having to maintain a complex autoexec.sas file or using %include.

X statements

The X statement in SAS allows the program to execute an operating system specific command from within a SAS program. This is one thing that will most likely break if we try to execute this in another environment. An example might be to get a directory listing. In Windows, we could get a directory listing (c:\temp) from SAS by issuing the following command from within SAS:

```
x 'dir c:\temp';
```

The same command when executed on CMS, MVS, OpenVMS, and UNIX will fail. In UNIX, we may have an alias set up which maps the

ls command to dir, but the directory path c:\temp will return a message telling you:

```
c:\temp not found.
```

If you know some sort of system command must be invoked, I highly suggest modularizing the program to segment the functionality into either a macro or an SCL call. In this manner, only one modification has to be made to change the functionality for another platform.

We can also avoid using the x statement by examining the %sysexec macro program statement, the call system routine, or the %sysget macro and sysget functions. Again, these should be used with caution when planning for a cross-platform port.

Automatic Macro Variables

The SAS System has several automatic macro variables that are instantiated at run-time. A few of the macro variables that are useful for cross-platform development include:

Macro Variable	Description
%SYSVER	Returns the version of the SAS System that is running (e.g. 6.12).
%SYSVLONG	Returns the version of the SAS System that is running but in a much longer form. (e.g. 6.12.0020P103196.)
%SYSSCP	Returns an abbreviation of the operating system in use (e.g. WIN, RS6000).
%SYSSCP1	Returns the full name of the operating system or environment in use (e.g. WIN 95, AIX).
%SYSJOBID	Returns a number that uniquely identifies the SAS task or the userid under which the program was run depending upon the operating system (e.g. MVS, CMS, VMS).

Version information

Version information can be extremely useful when we want to take advantage of a new feature in SAS, but yet want some level of backward compatibility in our application. The sysver and sysvlong macro variables can be used to determine if we are running the correct version of the SAS System to use a particular feature. For example, tooltips in SAS/AF® FRAME only became available in SAS Release 6.11, Wave 2. The following code shows an example of ascertaining the version in an SCL program.

```
release=symget('SYSVLONG');

if release le '6.11.0020' then

return;
```

Operating System

Detecting the current operating system programmatically can be very useful. The %syscp and %syscp1 automatic macro variables help tremendously by identifying either the operating system (%syscp) or the operating environment (%syscp1). Below is a good example of how we can use these macro variables if we know we are porting to another environment, or perhaps if the application is going to be deployed on multiple environments.

Macro language

```
%macro start;

  %if &sysscp= WIN %then %do;

    filename indata 'c:\temp\mydat.dat';

    libname mydata 'c:\mydata';

  %end;

  %else %if &sysscp= RS6000 %then %do;

    filename indata '/project/indata.dat';

    libname mydata '/project/mydata';

  %end;

%mend start;

%start
```

SCL

```
system=symget('SYSSCP');

select(system);

when('WIN')

  sysrc=filename('indata','c:\temp\mydata.dat');

  sysrc=libname('mydata','c:\mydata');

when('RS6000')

  sysrc=filename('indata','/project/indata.dat');

  sysrc=libname('mydata','/project');

end;
```

The following table shows the values of `sysscp` and `sysscpl` for the current SAS supported platforms:

Category	Operating System	SYSSCP	SYSSCPL
INTEL-Based Computers	PC-DOS	PC-DOS	
	Windows 3.1 (Win 32s)	WIN	WIN_32S
	Windows 95	WIN	WIN_95
	Windows NT (Workstation)	WIN	WIN_NT
	Windows NT (Server)	WIN	WIN-NTSV
	OS/2	OS2	OS2

UNIX Workstations and Servers	AIX (RS6000)	RS6000	AIX
	Digital UNIX	ALXOSF	DEC OSFI
	HP700	DEVHOST	HP-UX
	HP 9000/ 300	HP 300	HP-UX
	HP 9000/800	HP 800	HP-UX
	Solaris I	SUN 4	SunOS
	Solaris II	SUN 4	Solaris
	Intel ABI	386 ABI	386 ABI
	MIPS ABI	MIPS ABI	MIPS ABI
DEC Midrange	ALPHA AXP under OpenVMS	VMS_AXP	VMS_AXP
	Vax VMS	VMS	VMS
	Ultrix	ULTRIX	
Apple Computer	Macintosh (68000)	MAC	MAC_M68
	Macintosh (Power PC)	MAC	MAC_MPP
Mainframes	MVS	OS	MVS
	VM/SP	CMS	VM/SP
	VM/XA	CMS	VM/XA
	VM/ESA	CMS	VM/ESA
	VSE	VSE	VSE

Security Information

A very powerful development tool is the ability to access information from the operating system and then use it in a SAS application. In MVS, VMS and CMS, the automatic macro variable `sysjobid` contains the userid of the person who is running the SAS program. The values returned are very different in MVS and CMS than VMS, and have to be parsed very differently. Care must be taken when using this to capture the userid on a second platform.

In UNIX and NT, we can take advantage of environment variables to capture this information. In Windows 3.1, it is stored in an *.ini file. There are no automated methods for extracting this information into a SAS macro variable, however SAS can use such facilities such as `%sysget` macro to obtain the value of an environment variable like the current userid in UNIX (\$USER).

```
%let user =%sysget(USER);
```

```
%put &user;
```

Again, caution must be used when developing special procedures to capture the userid as these procedures may not be portable.

Products Installed

Although it is not an automatic macro variable, the `sysprod` function can be useful in determining whether or not a product is licensed on a particular platform. For example,

```
x=sysprod('graph');
```

The value returned is 1 if SAS/GRAPH software is currently licensed. The value returned is 0 if SAS/GRAPH software is not currently licensed. The `sysprod` function can be used in the DATA step, in an IML step, or in an SCL program.

Directory Based Systems

If we know that our application will be utilized across both Windows (Win 3.1, NT, 95) environments and UNIX file systems, we can use another trick to choose between the slash and the backslash if the naming conventions for our directories are consistent. For example, we would use the following macro to automatically determine which naming convention to use at run-time.

```
%macro system;

    %if %sysscp = RS6000 %then

        %let yy=;/

    %else %if %sysscp = WIN %then

        %let yy=\\;

    %mend system;

%system
```

If we were to submit the following `libname` statement, we would get different paths depending upon what system we were running under:

```
libname project "&yy.project&yy.clients";
```

In Windows, we would get:

```
\\project\clients
```

In UNIX (RS6000), we would get:

```
/project/clients
```

However, an undocumented feature in the SAS System for Microsoft Windows environments, we can use forward slashes in both `libname` and `filename` statements and they will resolve as if they were backslashes. For example, the `libname` statements below are identical in their resolution when running in a Microsoft Windows environment:

```
Libname test
'\projects\clients\myapp\macros';

Libname test
'/projects/clients/myapp/macros';
```

Both of these resolve to:

```
1 Libname test
'\projects\clients\myapp\macros';
```

NOTE: Libref TEST was successfully assigned as follows:

Engine: V612

Physical Name:
C:\projects\clients\myapp\macros

```
2 Libname test
'/projects/clients/myapp/macros';
```

NOTE: Libref TEST was successfully assigned as follows:

Engine: V612

Physical Name:
C:\projects\clients\myapp\macros

Case Sensitivity

An additional consideration in developing for multiple platforms is whether or not the file system is case sensitive. That is, are the file and directory names accessed the same way regardless of the capitalization of the letters contained in the file or directory names. In UNIX, the following files are different:

Myfile.txt	myfile.txt	MYFILE.TXT
------------	------------	------------

Even though these filenames are spelled the same, the capitalization makes them unique in a UNIX environment. In a Windows environment, each of these representations would refer to the same file - Windows is case-insensitive.

Host Features

Up to this point we've discussed some ways in which we can code our SAS programs and AF applications so that we can easily port them from one environment to another. SAS is a feature-rich environment, especially when we look at how it takes advantage of the operating system. In Windows, for example, we have a rich set of Windows-compliant application interface tools that allow us to communicate with the operating system and other applications.

It is to these operating system specific features that I would now like to turn. This is not intended to be an exhaustive listing of features related to each individual operating environment, but rather a note to the reader to be careful when planning to migrate applications to additional environments.

For an up-to-date list which identifies the environments/ platforms on which the SAS System is supported, please refer to: (<http://www.sas.com/software/tech/hardware2.html>).

Special Features/ Options

The best resource for the SAS programmer in determining important features of each operating system as they relate to the SAS System is the SAS Companion for the operating system being used. Here, we examine certain functions, invocation options, and conventions that we should take particular note of.

GUI Standards

Some environments support a graphical user interface (GUI) and others do not. Because of SAS's Multi-Vendor Architecture, we have the ability to develop in one environment and deploy the application in another. However *ideal* this may seem, we must make sure that we adhere to the GUI standards that users on those systems can live with. Users who use Microsoft Windows 95 as their standard desktop environment have come to expect a lot out of their applications. These

expectations are very different from those of a Macintosh user. If we were to put an AF application developed on MVS in front of either a seasoned Win 95 or Mac user and try to pass it off as a finished product, we'd probably have a mutiny on our hands.

James Hefner in his SUGI 19 paper (Hefner, 1994) gave an excellent resource for those of us wanting to port our application across multiple graphical environments. He shows us how many of the widgets we use in interactive applications will look in those environments. I strongly encourage developing a test application of not more than one or two FRAMEs and then seeing how it is going to look before further development. This would eliminate the dreaded possibility of having to revisit/redesign multiple screens or an entire application.

GUI and Mouse Support

As desktop environments like OS/2, Macintosh and Microsoft Windows become the standard for most of us, we get used to pointing-and-clicking our way around the screen. Even though most environments do have the ability to emulate graphical environments (e.g., X-windows, 3278-graphics emulation, etc.), we should be advised that users in these environments may not choose to take advantage of that capability. Target platforms should be checked for mouse and graphical support before developing to these standards.

Object Based Standards

Within SAS, we now have the ability to embed objects from other application environments. A great example recently noted was Jennifer Clegg and Carol Rigsbee's SUGI 21 paper where they described using an OLE object like a web browser within a FRAME application. SAS currently has a wide-range of support for integration with other applications. In SAS Communications (4th Qtr. 1994, 3rd Qtr. 1995 and 1st Qtr. 1996), the Institute announced the availability and continued support for:

- OLE
- DDE
- OLE Controls (OCX's and now ActiveX)
- ODBC
- E-mail (Microsoft MAPI and Lotus VIM)
- Lotus Notes

In future releases of SAS, we will probably see support for additional object models such as OpenDoc, CORBA, Taligent, NextStep and D-COM. As these object models take root in our SAS programs, we will need to ensure portability at a new level that we really didn't have to worry about before.

Help Systems

Writing help systems to guide our users through our applications were easy in the days of CBT entries. CBT (Computer Based Training) entries were stored in SAS catalogs and thus ran seamlessly on the myriad of platforms that SAS ran on. Building CBT entries could however be labor intensive, especially if we had to write end-user documentation in addition to providing the on-line help support. Tools like WinHelp, RoboHelp and the explosion of HTML as a standard format for documentation and help systems has made the development of on-line and user documentation a bit easier. HTML viewers are supported in graphical environments like Macintosh, OS/2 Windows, and UNIX as well as non-graphical environments with line-mode browsers like Lynx.

John Kruth, in his SUGI 22 paper (Kruth, 1997) talks about some new features in SAS/AF that help developers maintain help systems in the native environment (e.g., Windows, OS/2 and UNIX). There is however no magic cookie that will make all the ills go away. On one end, if we want complete portability, we should still use CBT entries; on the other, if we want enhanced platform-specific aids, we should use native help systems. Tools outside of SAS are getting better to assist us. For example, RoboHelp by Blue Sky Software (<http://www.blue-sky.com>) will aid in developing help screens for Windows, Windows 3.1 and Window 95 in native format, HTML, and printed documentation, as well as in generating web sites.

Printing/ E-mail/ Fax

In applications that we use, we take for granted that when we select the print function, it will work. As developers, we know that it is not that simple. This is particularly true when we develop applications in SAS for multiple environments. With version 6.12, SAS gave us some nice tools to help determine which printer is selected. Also in 6.12, host printing is available for SAS/GRAPH (except for Alpha/VMS hosts). Host printing permits sending the contents of the active window to the printer.

We can control the printer dialog box for the host by using Display Manager commands. The following table lists those available for managing printed output.

Command	Description
DLGPRT	Opens the Print dialog
DLGPRTPREVIEW	Invokes the Print Preview window

For example, the following SCL code controls the printer settings without the user having to specify anything.

```
call execcmdi('DLGPRTSETUP ORIENT=LANDSCAPE
NODISPLAY');
```

We can also get the current settings of the printer based on the system settings dialog box for the printer (again, without forcing the user to set these parameters.) In this example, we get the current settings and then tell SAS to use these as our DEVICE.

```
device=scan(optgetc('SYSPRINT'),1,' ');
rc=filename('DEVREF',device,'PRINTER');

... more code, then print

call notify('graph','_print_', ' ');
rc=filename('DEVREF','');
```

We could have simplified the above code, by using the following statement:

```
rc=filename('DEVREF','');
```

Note that the filename statement (in SCL) was passed a missing value. By default in 6.12, this will use the current system defined printer.

Printing can make or break a system for the user community. We shouldn't forget to test the printing facility on the platforms in which we are going to deploy our application.

Communications

The communications infrastructure that we rely on is pretty much taken for granted except when we need it. This is particularly true when we are developing systems off-site. Using a stand-alone machine to develop applications on and then redistributing them to a networked computing resource can be a fairly straightforward process. Something to keep in mind, however, is what communications protocols are available on the resource.

For example, developing an application under Windows and planning to port it to a UNIX environment sounds pretty straightforward. Knowing where the data reside in reference to the UNIX box and how users are planning to get to the application are things we must consider. If, in our example, all of the users were mounting a shared directory using NFS and using an X-windows graphical interface, we should be ok. However, if they plan to telnet into the system across the Internet in line-mode, our application interface would be useless if we used graphical FRAME objects.

In a recent data-warehousing project, I developed an extraction routine on a UNIX machine that extracted data from an Oracle database. The UNIX machine did not have Oracle, but rather used SQL*Net as the transport layer for the communication to Oracle. In this case, particular attention was paid to verifying that the SQL*Net connection was established. Had the Oracle database been resident on the host, that checking would not have been necessary. This serves to show us how important it is to know what some of the issues might be in dealing with the networking and transport layer of an application.

Performance

The scope of this paper doesn't lend itself to talking about performance beyond these few statements. When making decisions about migrating an application or building a new application, people tend to be concerned about "How it will perform?" What this really means is "It needs to be fast for the people who use it." Someone recently told me of a situation where a job ran in 4 hours on an MVS system. After the successful port to a UNIX system, the person was shocked to find out the job ran in 8 hours. Two morals to this story come to mind: First, run your benchmarks prior to migrating your entire system to test the feasibility of the port, and secondly, make sure that you are comparing apples to apples. MVS is a great environment for I/O intensive applications. Lots of reads and writes to disk and tape are common for MVS. This doesn't mean that UNIX is not good, it just means that we need to consider how our application uses resources on the host.

It turned out that the 4 hours was acceptable on the mainframe as the queuing system was good and there were not nearly the number of interactive users to notice the system degradation. After the port was made to UNIX, the programmer went through the code and optimized the SAS portion. Instead of simply using lots of `if-then` statements, he re-wrote them into `if-then/else` statements. He also paid more attention to how often the data needed to be sorted, and subsetting the data earlier in the cycle. There are reams and reams of articles and books on efficient programming techniques both in the SAS literature and in general computing. On MVS, the programmer was focusing on human-resource efficiency and in the UNIX environment he was forced to consider efficient processing of the computer.

Load-leveling and Multiple Processors

Mainframe systems have traditionally been very good at managing the allocation of system resources. When considering migrating an application from an MVS system to another environment we may want to consider how that environment determines priorities for both users and system resources. We probably don't want our researchers taking up all of the CPU when it's time to run monthly payroll.

Even though we are still in the infancy of multi-processor and shared memory systems, it's not too early to think about how we might want to balance the load between lots of users and system resources. Currently, SAS can support up to four processors on an NT server. In addition, SAS has developed a new Scalable Performance Data Server (SPDS). This multi-user server, which can be used to store SAS data sets, views, and catalogs, will be offered initially on Sun's SMP-enabled hardware and will take advantage of the Solaris threading package.

Data Storage and Access Mechanisms

Earlier we talked about some coding conventions to help get away from hard-coding things like `libname` and `filename` statements. There are several techniques we can use to help us in our cross-platform development, which should minimize the potential problems.

Source Management

Like most SAS programmers, I tend to store my programs as `*.sas` files on my operating system. They tend to be easier to find, edit and manipulate in either the SAS Display Manager System (DMS) or in an external editor like the SAS Viewer shipped with the desktop versions of SAS. If I later decide to move my application from one platform to another, I then have to be concerned with moving both binary files (like SAS datasets and catalogs) as well as text files containing SAS source code. One strategy might be to store these in SAS catalogs as `SOURCE` entries.

`SOURCE` entries can be transported from one environment to another by simply transferring the entire catalog - a useful feature if we have to maintain hundreds of SAS programs in addition to our data. In addition, `SOURCE` entries can retain attributes about the source such as color information that might be useful. I have one application where I used SAS/AF FRAME to front-end a series of macro calls. I didn't want the users to have to worry about what the appropriate macro parameters were, so I gave them selection lists, etc. to drive the macros. As a feature of this system, I also wanted to be able to give them full read access to the actual macro source code so they could see what was happening behind the scenes. By using standard naming conventions where the FRAME entry was the same as the `SOURCE` entry, the maintenance of this was pretty simply. When the user wanted to see the source code for the macro, I simply grabbed the name of the FRAME entry they were in and used the catalog viewer in FRAME to display the contents of the `SOURCE` entry in another generic FRAME.

Had I needed to port this application to another platform, I would simply use `PROC CPORT` to create a transport file, FTP the file to the new environment, and then use `PROC CIMPORT` to create the new library on my host. No changes to the SAS code would be needed outside of the `autoexec.sas` program.

Data access

Developing an application in one environment and then moving it back to another can be very frustrating - especially when we want to test our application with "real" data. As technology has advanced, the use of optical drives, CD-ROM, and removable hard-drives (like Iomega's Jaz drive) make it easy to take data off-site. We can build the application with larger sets of data instead of having to rely on error-prone techniques to sample our data.

Another technology that we have recently explored is Samba. Samba is a service that runs on a UNIX machine that allows the UNIX file system to be made available to a Windows networking environment. We've used this technology in a couple of projects where the users did not have SAS/CONNECT, but we wanted to make available a SAS dataset for them on their local system. The speed and performance of data access via Samba was good and at least one alternative to SAS/CONNECT for accessing remote data.

Relational Data Base Systems

Many vendors are developing what they call "personal" versions of their applications which have much of the functionality of the larger systems on which they are based. Recently, I had a project where I was able to replicate an entire Oracle database on my laptop using Personal Oracle. I took the data back with me and was able to write an application around the entire database. It allowed me to develop in virtually the same environment as it would be deployed. I also happened to have SAS/ACCESS to Oracle on my Windows 95 machine and was therefore able to write the SQL pass-thru code without having to worry about what it would look like when I installed the application. The benefit was that this allowed that part of the development to be done off-site.

ODBC

SAS/ACCESS to ODBC provides an interface to ODBC data sources. ODBC is a standard architecture for getting data out of proprietary systems. In the previous example, I talked about accessing Personal Oracle from within SAS with the SAS/ACCESS to Oracle product. Had I not had that access engine, I could have used SAS/ACCESS to ODBC to get the data out of Oracle just as easily, complete with the SQL pass-thru facility. In my case, I wrote a macro that enabled me to keep the parameters pretty much identical so that if someone didn't have SAS/Access to Oracle, they could use the same macro call to get the ODBC data. The macro calls are virtually identical:

```
%readdbms( dbms=oracle, userid=demo,
password=demo, intable=orders, outtable=TEST,
mpath=)
```

```
%readdbms( dbms=odbc, userid=, password=,
intable=orders, outtable=TEST, mpath=access)
```

Virtual Private Networks and the Internet

Sometimes data may change daily, so extracting the data and running off into the corner to do our programming just won't work. In these cases, we may have to use our telecommuting strategies to get the data. In SAS, we have additional access methods available on the filename statement to get data from remote networks. The URL and FTP access methods allow us to specify where the file is (a Web or FTP server) and authentication parameters to get the file. Another trick if we are developing an application on a machine that doesn't have a particular drive mapping that the end-users will have is to take advantage of technologies like RAS (remote access services) on Windows NT or point-to-point-tunnelling protocols (PPTP) to emulate our existence on that network.

Output management

A common task within an application is to store the results of an analysis program or query in a file on the host. This is definitely one of those things that will have to be re-written for another platform. If we can, we should try to write files to SAS catalogs. You have all the control in the world when reading and writing to SAS catalogs within both DATA step programming as well as SCL. An example of this might be if a user updated a customer database and wanted to write a note about that transaction. The note could very well be a SOURCE entry in a catalog. The name of the entry could be generated through knowing the customer's ID and transaction number. Both access and storage are now portable functions as they lie completely within the SAS System.

Design Considerations

When developing an application, we typically care about the system on which we are developing. If it looks good on a 3278 graphics terminal

and the users will access it through that interface, we should be OK. However, there are many design considerations that may not seem relevant until we want to take that application and move it to another system or display it via another mechanism. The best work that we've seen on advice for users is from a SAS Observations article (Vol 2, No 2) by Schlotzhauer and Thompson. Here I will simply review some of their recommendations.

Screen size

Lowest common denominators should when designing screen size in terms of both resolution (640 * 480 vs 1024 * 768) as well as the number of rows that can be displayed on a screen. If a data entry screen is developed in a higher resolution or one that supports more rows, we could find part of our data, or even part of our screen missing.

Color

In the desktop environments we have the ability to specify any one of more than 32,000 colors for properties of visuals widgets. Care should always be taken to ensure that those colors are available on the host that the application may be ported to. Colors used should be consistent. For example, red should be used to indicate an error and yellow for a warning.

Fonts

We can easily get caught up in using different font styles for our interactive applications. Anything looks better than the default DMS font, right! But be careful. Recently, I had to spend a few days going through an application because I specified a nice font that I had on my computer only to find out that the user did not have the font. SAS does try to remap the font into something it can find, but unfortunately, much of the text was cut off for extended text entries. Moral: standard fonts (ones that users will likely have available) should be used. A SAS System option can also be used when creating AF application that allows the use of what are called portable fonts. These fonts are available in one form or another on all platforms. To see these portable fonts in font selectors at build time, the following option can be specified in an autoexec.sas file.

```
options multenvappl;
```

Porting

Version Compatibility

When we port an application from one version of SAS to a previous version, we may run into problems. For example, in AF, a FRAME that was developed in version 6.09 is not downwardly compatible. If we need to take a 6.09 AF/FRAME application and run it in 6.08, we will need to use PROC CPORT with the V608 keyword on the procedure. With this option, CPORT will remove features that are not supported by an earlier release and rewrite the entry in an earlier internal entry structure. After running PROC CIMPORT on the new host, we will need to recompile the AF application.

As with any software development environment, there are features that are released with every new version that were not previously supported. For example, in SAS 6.09e and 6.11, there were a myriad of macro enhancements. If we use these and expect to be backward compatible, we can forget it. If this is a concern the sysver automatic macro variable can be used to code around the enhancement. The CPORT/CIMPORT procedure only supports the direction of movement from lower to higher releases, 6.09 to 6.10. Movement from Release 6.10 to 6.09 is not supported and may fail.

Binary Compatibility

SAS datasets, catalogs and views are not necessarily compatible from one platform to another in their native binary format. In addition, there are facilities within SAS that must be recompiled when transferred to another host. Stored programs and macro libraries are examples of SAS files that must be recompiled on the new host. In his SUGI 22 paper, Jeff Polzin (Polzin & Garner, 1997) wrote about the future of the binary compatibility problem. There he talked about a special engine type that would allow us to read a SAS file even though it was created on another host. For example, if I wanted to read a SAS dataset that was created on Windows, but was transferred to UNIX, I could use the following syntax:

```
libname windata sassfio '/winnt/sales.sd2';
```

The following table represents which files are compatible from one platform to another. The table shows whether we need to convert a UNIX SAS DATA file to a TRANSPORT file in order to use it on a different UNIX architecture, or whether we can use it as a SAS DATA file (not a SAS catalog).

Note: use means either physically move it (tape, etc. - not FTP) or remotely access via NFS, AFS, etc.

	Con vex- OS	HP- UX	Sun4	Solaris2	AIX	Ultrix	Windows	OS/2	Macintosh	MIPS ABI
Convex-OS	C	C	C	C	C	T	T	T	C	C
HP-UX	C	C	C	C	C	T	T	T	C	C
Sun4	C	C	C	C	C	T	T	T	C	C
Solaris2	C	C	C	C	C	T	T	T	C	C
AIX	C	C	C	C	C	T	T	T	C	C
Ultrix	T	T	T	T	T	C	T	T	T	T
Windows	T	T	T	T	T	T	C	T	T	T
OS/2	T	T	T	T	T	T	C	T	T	T
Macintosh	T	C	C	C	T	T	T	T	C	C
MIPS ABI	C	C	C	C	C	T	T	T	C	C

C means that SAS DATA sets and their index files can be read without problem (i.e. TRANSPORT formats are not required)

T means that SAS files should be converted to TRANSPORT format in order to be accessible by other platforms.

NOTE: Ultrix has a byte swapped architecture like INTEL.

The data set extension and the SAS release number can be used on a 68K Mac and/or a PowerMAC to assist in grouping UNIX compatible platforms. The MIPS ABI platforms like SGI are running Release 6.10 and have the .ssd01 extension. Data sets can be interchanged between these two compatible platforms without using a transport file.

HP, SUN (SUN4 and SOLARIS2), and AIX are running Release 6.09 and have the extension .ssd01. The data sets on these platforms are compatible with the 68K and PowerMAC and could be used without conversions via transport files.

SAS Catalogs

You must convert all catalogs to TRANSPORT format for moving between UNIX platforms, with exception of HP, Solaris1 and Solaris2 hosts, which can use the same catalogs without conversion. For all other platforms, the catalogs will need to be put into transport files using PROC CPORT/PROC CIMPORT.

File Transfer

Moving Text Files

If a text file such as a SAS program has to be moved from one account to another, or from one computer to another, FTP (File Transfer Protocol) should be used. To use FTP, account names and passwords for both the transfer-from and the transfer-to accounts must be known. The steps are as follow: log in to the computer from which we are transferring files; type ftp in response to the UNIX prompt; from the ftp> command line, type open HOSTNAME of the account to which files are being transferred (e.g. open power); enter the login name and password of the destination; at the ftp prompt, type put FILENAME (e.g., put mydata.ssd01); a reply will appear such as:

1191 bytes sent in 0.03 seconds (39 Kbytes/s)

When the process is complete, type quit. Now the transferred file exists in both accounts.

Moving System Files (SAS Library Files or SAS Datasets)

Occasionally, the need may arise to move a SAS System file from one computer system to another. For example, data may be received from a location which uses an operating system other than UNIX, or we may want to transfer files between SAS on a PC and SAS on a UNIX box. Since the form of SAS System files is specific to the operating system under which the files have been created, moving system files from one machine to the other requires some extra steps. Note that if a system file is being moved from one UNIX account to another, export and import processes are not needed - UNIX system files (data sets and some catalogs) may be easily transferred from one UNIX box to another.

Using SAS/Connect

Using SAS/Connect to transfer SAS datasets, catalogs and even libraries can be fairly straightforward. This assumes that a SAS/CONNECT session can be established with the system being transferred to/from. Once this connection is established via a connection script, using SAS/CONNECT to communicate with another host is very simple. (Sample scripts are distributed with SAS software in sasroot/connect/saslink for various platforms.)

The following code is what I submit to log on to an RS-6000 and start a SAS/CONNECT session between the UNIX machine and Windows 95.

```
options comamid=tcp remote=apollo;

filename rlink 'h:\myhost.scr';

signon;
```

Once this code is submitted, a connection will exist to the remote host (UNIX). Information can now be transferred from the local host (Windows 95) machine to the UNIX machine.

Uploading a SAS Dataset

To upload a SAS dataset, the "local" dataset is specified with the data= option and the "remote" dataset with the out= option, as in:

```
rsubmit apollo;

libname unixlib '/project';

proc upload data=sasuser.crime
    out=unixlib.crime;

quit;

endrsubmit;
```

Uploading a SAS Catalog

When uploading a SAS catalog, the "local" catalog is specified with the incat= option and the "remote" catalog is specified with outcat= option, as in:

```
rsubmit apollo;

libname unixlib '/project';

proc upload incat=sasuser.windows
    outcat=unixlib.windows;

quit;

endrsubmit;
```

Uploading an entire SAS Library

Often, when porting an application, it is useful to take an entire library complete with catalog entries and datasets and upload them to you're the remote host. We can do this by specifying the inlib= and outlib= option on the PROC UPLOAD procedure, as in:

```
rsubmit apollo;

libname unixlib '/project';

proc upload inlib=sasuser outlib=unixlib;

quit;

endrsubmit;
```

Of course there are lots of options we can specify when using PROC UPLOAD that allows us to specify particular type of entries (entrytype=), or specific files (select statement).

Using FTP

If SAS/CONNECT is not available for transferring files, other transport mechanism such as Kermit or FTP (File Transfer Protocol) can be used. The following shows how to transfer the information via FTP. First, the system file must be converted to a portable file by running the short SAS export program shown below. Note here I use PROC COPY, but alternatively, PROC CPORT and PROC CIMPORT may be used. Portable files are versions of system files that can be imported into SAS under all operating systems. Second, FTP is used to move the portable file to the destination computer. It is important to remember that the binary FTP format must be used to move the file. (See the example on page 10.) Third, the portable file is imported into a standard system file on the destination computer by running another short SAS program.

Once this has all been done, the portable transport file can be erased (from both accounts). (This erasing should only be done AFTER checking that the data has been imported correctly (for example, by using the PRINT or CONTENTS procedures).

Note: It may be necessary to move a directory that consists of several system files at once. In the example below, we move a directory called sasfiles by writing it into a portable file called expfile.exp that can be FTP'd and then imported. Expfile.exp contains a portable version of all the system files in the sasfiles directory. Before exporting, we make sure that the file or files we are about to export are in a separate directory. Here are the steps:

1. We create the following export program and run it.

```
libname mylib '~/stat/sasfiles';

libname tranfile XPORT 'expfile.exp';

proc copy in = mylib out = tranfile;

run;
```

If we type ls at the system prompt, we would see that we now have a new file called expfile.exp.

2. We FTP the file expfile.exp (following the instructions from the earlier discussion of FTP for text files, but this time we type binary at the ftp> command line and press the return key before entering the put command.

3. On the destination computer, we create and run the following import program, which saves the data from the portable file back into all the standard system files from sasfiles.

```
libname tranfile xport 'expfile.exp';

libname newlib '~/usinfo/new';

proc copy in = tranfile out = newlib;

run;
```

Client/ Server Issues

Throughout this paper, we have focused on the migration or deployment of an application from one environment to a completely different platform. Although this may be a typical scenario, another situation exists that deserves comment. If an application is running on a mainframe or UNIX machine and we want to convert it to a Windows application, the best solution is not always to simply move the application to the local desktop platform. This is where our skills as a developer will definitely be tested.

One of the biggest challenges in developing true client/ server applications is knowing where things happen. In a stand-alone environment where the application logic, computing resources and data reside on the same system, this is not an issue. However, in distributed applications we may run our application on a Windows platform which accesses data on a remote host like a UNIX server. Even more complicated scenarios are common. For example, I wrote a Windows SAS/AF application which accessed local data resources (NT Server), data that was on a UNIX machine and transaction data on an MVS system. I also allowed the users to transparently use any of the three resources for compute services. If the datasets were large or the analysis required compute intensive resources, the code was submitted on the UNIX machine. In these cases, there aren't many tools for SAS programmers to utilize which makes this transparent. SAS/CONNECT obviously makes this doable, but the programming logic was complex.

One suggestion that I would make here is to take advantage of macro variables at run-time to decide where requests should be processed. For example, in SCL, we often use submit blocks to submit sections of code to the SAS System for processing. By using a macro variable, we can simply change the submit block to a remote submit block. Going through you're a large piece of SCL code to change all the submits to rsubmits will show just how useful this little technique can be.

Conclusion

The goal of this paper was to provide the reader with a practical guide to cross-platform applications development. We've discussed in brief the merits of Multi-Vendor Architecture™ and gave some practical solutions to common issues in multi-platform deployment. As we have stated before, there is no right-way to build an application within the SAS System, but there are some fairly simple ways which will make migration and client/ server conversions a bit easier.

One of the clear strengths of the SAS System is the ability to deploy a SAS application on a multitude of systems. As SAS Institute takes advantage of new technologies on specific platforms, we must be careful to ensure the portability of these applications.

Acknowledgements

I would like to acknowledge the assistance of ASG, Inc., my employer, for providing me the time and resources for developing this paper. Also, a special thanks to Ray Pass, Cherry Dunn and Richard D. Langston for their assistance in reviewing this manuscript.

Bibliography

- Meeting the Challenge: A New Generation of Software for the 90's.* SAS Communications, Vol. XV, No. 3. 1st Quarter, 1990, pp. 3-8.
- Providing a Solid Foundation for Interoperability.* SAS Communications, Vol. XX, No.4, 4th qtr, 1994.

No Software is an Island. SAS Communications. Vol. XXI, No. 3, 3rd qtr, 1995.

The Open Door to Applications Integration. SAS Communications. Vol. XXII, No. 1, 1st qtr, 1996.

Clegg, J. & Riggsbee, C. (1996). *OLE and the SAS System for Windows, Release 6.12*. Published in the proceedings of the 22th Annual Conference of the International SAS Users Group - San Diego, CA. Pp. 257-266.

Hefner, J. (1994) *This is a Fairy Tale... NOT! A Primer on Moving SAS Applications Across Graphical Operating Systems*. Published in the proceedings of the 19th Annual Conference of the International SAS Users Group - Dallas, TX. Pp. 404-409.

Kruth, J. (1997). *Native Help Technology in SAS/AF Applications*. Published in the proceedings of the 22th Annual Conference of the International SAS Users Group - San Diego, CA. Pp. 1520-1523.

Polzin, J. & Garner, C. (1997). *Cross Platform Access to SAS Data Files*. Published in the proceedings of the 22th Annual Conference of the International SAS Users Group - San Diego, CA. (available only on Web Site).

Schlotzhauer, S.D. & Thompson, E. T. (1993) *Ensuring Portability of SAS/AF Applications*. *Observations*. Vol. 2, No. 2. 1st Qtr.

Address

Questions pertaining to this article should be addressed to:

Gregory S. Barnes Nelson
ASG, Inc.
2000 Regency Parkway, Suite 355
Cary, NC 27511
919-467-0505 (Voice)
919-467-2469 (Fax)
gbn@asg-inc.com