Paper 1-28

Multi-platform SAS®, Multi-platform code

David Johnson, DKV-J Consultancies, Holmeswood, England

ABSTRACT

Many people who write SAS code are likely to find that programs they have written for one platform will need to be migrated to another. For a consultant, this is a frequent occurrence. However, it is also a strong possibility for permanent staff, as many members of the SAS/L community have reported.

Whether it is to meet the need for programs to move to a new mid range server, decommissioning of an existing platform, or implementation of a new information delivery architecture, a platform change may be required for existing programs.

Platform independence is usually going to mean that macros are called in the program to perform certain operating system dependent tasks. Then the macro libraries can be ‘switched’ when the programs are moved, retaining the existing code, unchanged.

We’ll look at a number of Operating System specific processes for MVS, Unix and Windows, and discuss the ways these techniques have been implemented.

This paper assumes an introductory knowledge of macros. Knowledge of the different operating systems would be beneficial.

INTRODUCTION

The most recent experience of a platform move occurred where a client required existing programs be made more robust. The strategy of the client was to create production code that would run without intervention, could be a template for other development, and would migrate from the existing Windows platform to a Unix platform.

The second experience involved development of a query process for a data mart on a mainframe, where the data mart was not yet available. Model data marts were built on Windows, and the code written to deal with these data marts. When the mainframe data mart was deployed, the programs needed to be switched to run on MVS.

A third experience is a frequent requirement for a consultant. Common code modules are developed and held in the consultants ‘tool box’, ready to be deployed for a client. These tools standardise common tasks, and allow faster development of robust programs. In some cases, the development platform may be a mid-range or PC, with the production system running on a more robust (and expensive) platform.

The common thread with all these requirements was that parts of the code that performed ‘utility processes’ were to be handled with object based processes. Sometimes this assisted other less experienced developers in building robust code, but it also reduced the development and customisation time for all the developers.

We will look at a handful of processes for each of the OS/390, Unix and Windows platforms. We’ll discuss the way in which the macro was developed for the first platform, then look at how similar information is made available from the others. We’ll also look at how we handled information from a ‘richer’ platform when we developed a macro that had to function similarly to a macro developed on a less informative platform.

This paper assumes an introductory knowledge of macros. Knowledge of the different operating systems would be beneficial.

IT WASN'T DEVELOPED HERE

Sometimes, for resource, choice or availability reasons, code may need to be developed on one platform and run on another. The change freeze for Production platforms around year 2000 was a classic occasion where project development may not be stopped, but any change at all in the core reporting systems was an unacceptable risk.

It was this restriction that moved a clients' substantial data cleansing effort from the mainframe to the Warehouse' Unix platform. Access to the Unix platform was limited and used rudimentary tools, so most code development occurred on the Windows platform.

In this situation, development of the code to load our new data warehouse was prototyped and tested on a Windows platform, but was run on Unix. Since programs were transferred between the platforms and run locally, rather than through SAS/Connect, macros called within the program needed to exist and work on both platforms.

Many of the macros used calls to the Operating System. Those developed for Windows used the Windows API almost exclusively. Those run on Unix used Operating System commands submitted and captured through filename allocations using unnamed ‘Pipes’.

FINDING FILES IN WINDOWS

The Windows API to find files involves a number of smaller pieces, and is dealt with in detail when we look at the file find process for Windows. In summary though, once a file that matched a search criterion was located, a call to a series of APIs would surface details on the file parameters. These included:

- File modification date
- File attribute flags - all systems: R (Read Only), A (Archived), S (System) and H (Hidden)
- File attribute flags - Windows NT: C (Compressed).
- File creation date (not available Windows 95)
- File access date (not available Windows 95)

It was clear very early in the design process that not all information was available on all platforms. If a Windows macro was to be effective as a generic tool, then it must also deal with a number of different versions of the Windows Operating System.

FINDING FILES IN UNIX

In Unix, there was no API interface readily available. So the device type 'Pipe' was invoked on the Filename allocation. This allowed an Operating System command to be submitted, and its results surfaced to the SAS session. The Unix command 'ls -al' allowed adequate data to be made available on Unix files, but the parsing and validation of the text returned was too complex to be

undertaken on an ad-hoc basis. A macro was needed to deal with the data reading and validation on a consistent and robust basis. (A detailed examination of the Unix file find process appears in appendix 2.)

On comparison with the Windows return values, a number of issues arose. The file dates available related solely to the creation date of the file, and the file attributes were substantially different. Within Unix, a file has three possible actions. They are 'R' (Read), 'W' (Write) and 'X' (eXecute).

However, they are defined for three different types of user. The first relates to the file owner, the second to the group to which the file belongs and the third to the global environment. In this way a file may, for example, be updated by the owner, read by other members of the owners group and hidden from all other users on the machine.

BRINGING FILE INFORMATION TOGETHER

You can see that this presents some real problems to the user. If one distils the information surfaced by each environment (Windows 95, Windows NT and Unix) to the lowest common denominator, then valuable information is lost from each.

The solution was to find the common points of agreement, and name the variables for each Operating System similarly. Then additional variables were made available on the data set created to allow for the extra attributes. It was also essential that each macro was comprehensively documented, recording the points of comparison across Operating Systems, and providing warnings for those areas where differences would cause cross platform problems.

HOW DO THE MACROS COMPARE?

Looking more closely at the macros developed for Windows and Unix, we find that they have **common names and parameters**. This was the first and most important rule of development. To support this naming, all operating system macros were named with 'X' as the first character.

Where one Operating System requires an additional parameter not provided on another, the extra parameter was provided on both macros. It was assigned a default value on the macro that didn't populate it meaningfully. In this way, a macro that had been extensively used previously could be extended without threatening existing applications.

The data set built within the macro for file finding had a common name on both macros. To prevent issues of contention however, the **name of the data set always met a certain naming standard**. All began with the letters ZZ, and the site programming standards specified that no data set named with ZZ be to be created in any application. This was also most important, and one of the few times I have seen a SAS programming standard rigidly enforced.

Aside from the 'File Find' macros, many others were created, some providing a single value within a macro token. The third standard to be applied defined that all such macro tokens would begin with 'Z' and no code outside a System macro was to create any macro similarly named. This was the third standard to be rigidly applied.

A SIMPLE EXAMPLE

Since the application was being developed as a Production system, responsibility for reports and output generated from the system could change on a regular basis. The identification of the user running the SAS session that produced the output, would be the person responsible for following up issues. Differing approaches needed to be adopted depending on the platform. Let us look at how a solution for mainframe Production reporting jobs was applied to two other platforms.

REPORT NAMING - OS/390 PROGRAMS

For mainframe jobs, the identity of the person running the jobs can be retrieved in two ways. First, the SAS System macro symbol SysJobId will contain the name of the job, as identified on the job card. In Production, this is usually a standard user name, so we can compare this value against a database. This database could be populated on a regular basis from extracts generated from Access Control Files (ACF, RACF et al).

In the case of fully scheduled production jobs, we can substitute the name of the scheduler with the name of the person responsible ('Duty Analyst'). The process diagram looks something like that in Figure 1.

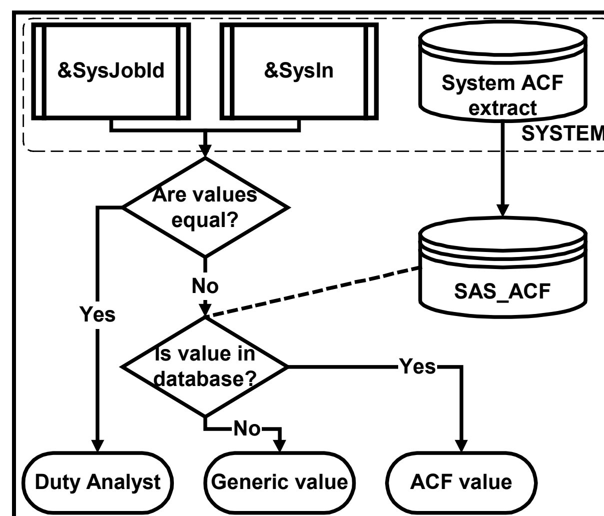


Figure 1

The approach proved to be robust, and the monthly reports were produced with footnote entries that identified the name of the person responsible, and the contact telephone number. All of the code was embedded in a macro called ***XGetUser***, which was distributed in the 'SASAUTOS' directory assigned to the SAS session at start-up. This macro also created the footnote for the report, which included the name of the SAS source code library entry (***SysIn***). In later releases, the macro was further extended to identify the email recipient for critical messaging.

The further benefit of this approach was that development programs were also written with a call to the macro. Since the site standard specified that certain macro values were 'reserved' for local processing rules, and that the first footnote would be provided by the macro, all output generated was automatically identified with the analyst responsible. We'll come back once more to the 'reserved' values after we look at the same approach for Unix and Windows.

A more recent development involved reading the data contained in the system pointers to the Task I/O system, the System Management Facility and the Job System Control Block. The approach involves quite a lot of code, so this has been provided in the first appendix, called “PEEKing and POKEing your mainframe.”

REPORT NAMING - UNIX PROGRAMS

The problem with using SysJobId with SAS on the Unix platform can be found in the 'SAS Companion for Unix environments'. It states: " (***SYSJOBID***) lists the PID of the process that is executing the SAS System, for example, 0024". The Process Identifier (PID) is derived from the Unix master list of processes running on the machine at the time of the start of the process. It bears no relation to the user who started the job.

To retrieve the name of the user, a different approach was needed. The approach involved sending a command to the Operating System, and then retrieving the result. The method used was to issue a SAS 'Filename' statement with the device option set to 'PIPE'. Then the command was embedded in the external file reference. Calling this file reference in a data step would cause the command to be issued, and surface the return value to the data step. Then this could be parsed and a value retrieved. The SAS code looks like this:

```
Filename  SYSREQST  Pipe  'env';
  Data _NULL_;
    Retain USERNAME 'unknown username';
    Infile SYSREQST  TruncOver;
    Input @1 READSTR $char80.;
    If READSTR Eq: "LOGNAME" Then  USERNAME =
       Trim( Substr( READSTR, Index(
          READSTR, '=')+1 ) );
    Call Symput( 'ZLogName', USERNAME);
Run;
```

This is an abbreviated version of the actual macro that was deployed. The final macro used the environment (env) command to also surface data on the current working directory, timezone, path assignment, mail path and terminal type for other system integrated functionality. It has also been tested on a number of Unices, including HP-UX and Sun Solaris. Some differences exist depending on the version, vendor and set-up, but similar sets of information can be retrieved.

This process was modelled at a site where the following rules were applied:

The production username for SAS applications was 'SASADMIN'.
All production SAS data was owned by SASADMIN.
The SASADMIN and all individual users of SAS applications were members of the user group 'SASUSERS'.
All data created by SASADMIN and all members of SASUSERS was readable by the group SASUSERS, and hidden from the general user population.

SAS data created by a Production run was then secured against unwanted modification by other users, and was accessible only to the group who had business with the data. To report on batch processes, the duty analyst' name appeared in the footnote, along with the project help line number.

REPORT NAMING - WINDOWS PROGRAMS

If Unix was difficult, Windows was even more complex. The 'SAS Companion for the Microsoft Windows Environment' tells us "(***SYSJOBID***) returns a number that uniquely identifies the SAS task under Windows". This is similar to the Unix process number, and equally uninformative. While some sites store user name information in Operating System tables or Windows environment variables, this is not done consistently.

The first point of consistency across most sites however, is that the Windows machine is usually connected to a Local Area Network. A LAN identity may then provide a unique user identifier. To use this value, we need to surface the LAN ID through a call to the Windows API.

The Windows API issues an executable command to the Operating System through an External DLL. These Dynamic Link Libraries are files that contain many commands or utilities that may be used by many applications. The process for using external DLLs is treated in some detail in a number of additional resources made available by SAS. These are listed at the end of the article and are very good supplementary resources.

The principle of using external DLLs is threefold:

- Create a 'prototype' for the command. This must contain the DLL name, the function name, and the attributes that need to be passed to the DLL.
- Save this prototype in a text file and make it available to the SAS session by using the reserved file name 'SASCBTBL'.
- Call the command by using the 'ModuleN' function with the appropriate number of parameters.

Here is a prototype for an API that will find the Network logon identity for a user when Windows networking is employed:

```
routine WNetGetUserA
  minarg   = 3
  maxarg   = 3
  stackpop = called
  returns  = long
  module   = mpr;
arg 1 char  update  format = $cstr20.;
arg 2 char  output  format = $cstr20.;
arg 3 num   update  format = pib4.;
```

Retrieving the value involves submitting code as we did in the following SAS session:

```
165  Data _NULL_;
166    Length DUMMY  USER $20.;
167    DUMMY = " ";
168    USER  = " ";
169    RC = ModuleN( "WNetGetUserA", DUMMY,
USER, 255);
170    If RC >= 1 Then Put @5
171     "Network & username not available.";
172    Else If RC < 0 Then Put @5
173     "Network not available or not
responding.";
174    Else Put 'Lan User name found : '
USER;
175  Run;

Lan User name found : Administrator
NOTE: The DATA statement used 0.34 seconds.
```

The flaw with this approach is that a network connection is required, and it must be either Windows networking or one of certain other Network types. It has not worked on some networks tested. As a backup to this, we can retrieve the name of the person who logged on to Windows. This is done with the GetUserNameA routine. The prototype for this routine follows.

```
routine GetUserNameA
  minarg   = 2
  maxarg   = 2
  stackpop = called
  module   = advapi32
  returns  = long;
arg 1 char  update  format = $cstr20.;
arg 2 num   update  format = pib4.;
```

By combining the calls to the two APIs, we can test for the second value if the first does not exist. The commented statement at line 255 below will prevent resetting the value if the WNetGetUserA call is successful.

```
244  Data _NULL_;
245    Length DUMMY  USER $20.;
246    DUMMY = " ";
```

```
247    USER   = " ";
248    RC = ModuleN( "WNetGetUserA", DUMMY,
USER, 255);
249    If RC >= 1 Then Put @5
250     "Network & username not available.";
251    Else If RC < 0 Then Put @5
252       "Network not available or not
responding.";
253    Else Put @5 'Lan User name found : '
USER;
254    Put 'SUGI debug WNetGetUserA ' RC=;
255    /* If RC = 1  Then Stop; */
256    RC = ModuleN("GetUserNameA",USER,255);
257    If RC = 1 Then Put @5
258     "Windows logon user " user "found.";
259    Else Do;
260      If RC > 1 Then Put @5
261         "Error in retrieving Windows
logon.";
262      Else Put @5
263         "Windows logon responding, no
username found.";
264    End;
265    Put 'SUGI debug GetUserNameA ' RC=;
266  Run;

    Lan User name found : Administrator
SUGI debug WNetGetUserA RC=0
    Windows logon user Administrator found.
SUGI debug GetUserNameA RC=1
NOTE: The DATA statement used 0.58 seconds.
```

The log entries marked 'SUGI Debug' were included to highlight one of the potential pitfalls in using APIs. The return code from each successful call may be different. This means that developing a Windows API call from scratch will require some careful research on your part into the API details.

I know of, and use three electronic resources to assist with the definition of the prototype in the attribute file, and the returns expected from each call. They are:

- the Windows Software Development Kit,
- the support area of the Microsoft web site and
- the WIN32.HLP help file packaged with Borland Delphi.

This last file was made available from Microsoft, and as such may not reflect recent changes to your operating system. However, for anyone else who has the product licensed, it is a very good resource. Here in the UK, Borland have generously made a number of releases of Delphi available on magazine cover disks, so the help file is fairly commonly available. I should also add that there are quite a few good books available dealing with the subject of Windows APIs. If you'd like some recommendations, feel free to write to the author.

The only significant question with the API approach is whether the user is logged on to the machine. In the case of Windows 3.1 and Windows 95, it is possible to start up the Operating System without logging on. In this case, the Windows macro needs to take account of the version of Windows. This can be retrieved with a Windows API call, and is also available through the automatic global macro tokens defined within the SAS session. The following is part of the entry put to a SAS log when the command '*%Put _all_;*' is submitted to a SAS session:

```
AUTOMATIC SYSSCPL WIN_NT
```

By reading the value of the ***SysSCpl*** token, we can include branches in our ***GetUser*** macro that accommodate different Windows platforms.

EMAILING RESULTS TO WHOM?

One of the more common requirements that emerged with production applications was the need to deliver information more quickly. The author has seen the increasing use of email as an information delivery tool in production applications. Aside from the clear benefits of delivering business information more quickly to the information users, there was also a strong benefit in delivering progress and failure reports quickly.

The ***GetUser*** macros were easily extended to provide an email address that could be used to report process continuation, completion or failure. Since each program on a given platform used a common macro, a small change to a macro could surface an email address to the program from a reference data set. Then those programs requiring email functionality could be extended to generate the appropriate messages.

ARE ALL WINDOWS APIS SO EASY TO SET UP?

The simple answer is 'No'. All the authors of API related information below warn against casual API experimentation. An error in an API call can crash a SAS session or a Windows session resulting in loss of data. One of the more involved macros undertaken by the author is the FileFind macro. It comprises one master macro, and two subsidiary ones. It also uses six API prototypes, which follow. (I have edited them for space reasons. The full versions are available on the author's web site.)

```
routine FindFirstFileA
        minarg=12
        maxarg=12
        stackpop=called
        module=Kernel32
        returns=long;
arg  1 char input  format=$cstr200.;
arg  2 num  update fdstart  format=pib4.;
arg  3 num  update format=pib8.;
arg  4 num  update format=pib8.;
arg  5 num  update format=pib8.;
arg  6 num  update format=pib4.;
arg  7 num  update format=pib4.;
arg  8 num  output format=pib4.;
arg  9 num  output format=pib4.;
arg 10 char update format=$CSTR200.;
arg 11 char update format=$CSTR60.;
arg 12 char update format=$CSTR14.;


routine FindNextFileA
        minarg=12
        maxarg=12
        stackpop=called
        module=Kernel32
        returns=long;
arg  1 num  input  byvalue  format=pib4;
< arg 2 - 12 of FindFirstFileA routine >

routine FindClose
  minarg=1
  maxarg=1
  stackpop=called
  module=kernel32
  returns=short;
arg 1 num input byvalue format=pib4.;


routine GetFileTime
  minarg=4
  maxarg=4
  stackpop=called
  module=kernel32
```

```
  returns=ulong;
arg 1 num update byvalue format=pib4.;
arg 2 num update format=pib8.;
arg 3 num update format=pib8.;
arg 4 num update format=pib8.;


routine FileTimeToLocalFileTime
  minarg=2
  maxarg=2
  stackpop=called
  module=kernel32
  returns=ulong;
arg 1 input  fdstart num format=pib8.;
arg 2 output fdstart num format=pib8.;


routine FileTimeToSystemTime
        minarg=9
        maxarg=9
        stackpop=called
        module=Kernel32
        returns=long;
arg  1 num  input          format=pib8.;
arg  2 num  output fdstart format=pib2.;
arg  3 num  output         format=pib2.;
arg  4 num  output         format=pib2.;
arg  5 num  output         format=pib2.;
arg  6 num  output         format=pib2.;
arg  7 num  output         format=pib2.;
arg  8 num  output         format=pib2.;
arg  9 num  output         format=pib2.;
```

The above attributes are referenced as needed in the following macro. This finds the first file matching the search criteria, and then uses that successful search to find each other file that also matches.

```
%Macro XFileFnd( MDir = ) /
   Des="SYSTEM: List MDIR to ZZZZFILE";

  Data ZZZZFILE( Keep = FILENAME  EXTENSN
                CREATESD  ACCESSSD  WRITESD
                FILEATT  FILESIZE);
    Length FILENAME NAME PATH $200
        EXTENSN $32  FILEATT $8  BITNAME $60
        ANAME $14  CREATESD ACCESSSD WRITESD
        FILESIZE 8;
    NAME = " ";  BITNAME = ' ';  ANAME = " ";
    ATT  = .;    CRE     = .;    ACC   = .;
    WRI  = .;    SLOW    = .;    SHIGH = .;
    RC   = .;    NUM1    = .;    NUM2  = .;
    LNUM1= .;    RCODE   = 0;
    PATH = "&MDir";
    RC   = ModuleN( 'SetLastError',0);
    HANDLE = ModuleN( 'FindFirstFileA',
         PATH, ATT, CRE, ACC, WRI, SHIGH,
         SLOW, 0, 0, NAME, BITNAME, ANAME);
    RC     = ModuleN( 'GetLastError');
    If RC Ne 2  Then Put 'WARNING: There is
         a problem with your API call.';
    If HANDLE >= 1  Then Do;
      %XGetAttr;
      Output;
      FOUND = 1;
      Do While( FOUND);
        FILEATT = ' ';
        FOUND = ModuleN( 'FindNextFileA',
           HANDLE, ATT, CRE, ACC, WRI, SHIGH,
           SLOW, 0, 0,NAME, BITNAME, ANAME);
        If FOUND   Then   Do;
          %XGetAttr;
          Output;
        End;
      End;
      RC = ModuleN( 'FindClose', HANDLE);
    End;
  Run;

%Mend XFileFnd;
```

For each located file, a call is made to the ***XGetAttr*** macro, which reads, interprets and validates the attributes of the file. The following is the ***XGetAttr*** macro.

```
%Macro XGetAttr /
    Des = 'Get file attributes';

  ** Reverse the string in case multiple
periods appear in the file name;
  If Length( Trim( NAME) ) > 2  And
     Index( NAME, '.')  Then Do;
    POSITION = Length( Trim( NAME) ) - Index(
        Left( Reverse( NAME) ), '.') + 1;
    FILENAME = Substr( NAME, 1,
                       POSITION - 1);
    EXTENSN  = Substr( NAME,
                       POSITION + 1);
  End;

  Else Do;
    FILENAME = NAME;
    EXTENSN  = ' ';
  End;

  %XGetTime( MFTime=CRE, MDSTime=CREATESD);
  %XGetTime( MFTime=ACC, MDSTime=ACCESSSD);
  %XGetTime( MFTime=WRI, MDSTime=WRITESD);

  FILESIZE = SHIGH * ( 2**32) + SLOW;
  If Input( Substr( Put( ATT, Binary8.),
           1, 1), 1.) Then ATT = 0;
  If Input( Substr( Put( ATT, Binary8.),
           4, 1), 1.) Then FILEATT = 'F';
  If Input( Substr( Put( ATT, Binary8.),
           8, 1), 1.) Then FILEATT =
        Compress( FILEATT || "R");
  If Input( Substr( Put( ATT, Binary8.),
           3, 1), 1.) Then FILEATT =
        Compress( FILEATT || "A");
  If Input( Substr( Put( ATT, Binary8.),
           6, 1), 1.) Then FILEATT =
        Compress( FILEATT || "S");
  If Input( Substr( Put( ATT, Binary8.),
           7, 1), 1.)  Then FILEATT =
        Compress( FILEATT || "H");

%Mend XGetAttr;
```

The ***XGetAttr*** macro has up to three date-time values to interpret. These three values represent the Creation date, the Modification date and the Last accessed date for the file. (Not all are available in all versions of Windows.) The three pointers returned by the file find processes are passed in turn into the ***XGetTime*** macro and valid dates resolved.

```
%Macro XGetTime( MFTime=, MDSTime=);

  &MDSTime = .;  YEAR = .;     MONTH = .;
  DAYOFWK = .;   DAY = .;      HOUR = .;
  MINUTE = .;    SECONDS = .; MSECONDS = .;
  If HANDLE >= 1  Then RC4 =
      ModuleN( 'FileTimeToLocalFileTime',
              &MFTime, LNUM1);
```

```
     If RC4  Then RC5 =
         ModuleN( 'FileTimeToSystemTime',
            LNUM1, YEAR, MONTH, DAYOFWK, DAY,
            HOUR, MINUTE, SECONDS, MSECONDS);
     If  YEAR > Year( Date() )     Or
         MONTH > 12      Or  DAYOFWK > 7   Or
         DAY > 31        Or  HOUR > 24     Or
         MINUTE > 60     Or  SECONDS > 60  Or
         MSECONDS > 999     Then    Put "WARNING:
            &MDSTime cannot be created because
            of faulty date time data" /
            @5  NAME =      YEAR =     MONTH =
                DAYOFWK =   DAY =      HOUR =
                MINUTE =    SECONDS =
                MSECONDS =;
     Else  &MDSTime =  Dhms(
           Mdy( MONTH, DAY, YEAR),
           HOUR, MINUTE, SECONDS +
            ( MSECONDS / 1000) );
     Format &MDSTime DateTime21.2;

   %Mend XGetTime;
```

The date time components are all independently validated. The author has seen the date time stamps of a large group of files corrupted by a malfunctioning application.

For those interested, a comparable macro for OS/390 follows in Appendix 3. It uses a utility for retrieving information from the Data set Management System (DMS). The utility is usually found in the 'System Proc' libraries on IBM mainframes.

APPENDIX 1: PEEKING AND POKEING YOUR MAINFRAME

The user details on a Job are somewhat esoteric, since they call for some less commonly used programming skills. Firstly, we retrieve information on the submitted job from the Task Control Block (TCB). Using a PEEK() function, we can retrieve a pointer to the Task I/O information, as well as the Job System Control Block (JSCB) and other valuable information.

The function directly reads values stored in the system memory, and should be used with great care. Attempting to read the wrong memory address can crash the system. This type of technique is well suited to encapsulating in a macro both because of its complexity and its sensitivity.

Some additional information is captured to the ZZGETUSR data set from standard SAS macros. The SAS System version is useful for verifying the release of the SAS version in Production reporting. The name of the Host System is useful for conditionally controlling programs based on the Operating System. On OS/390, all systems I have seen have reported this value as 'MVS'.

```
DATA ZZGETUSR( Drop = TIOTPTR  JSCBPTR  SSIBPTR
                      JCTPTR   ACTPTR
              Label = 'User information on MVS job');
  LENGTH PROGRAMR $20  JOBNAME $8   PROCSTEP $8
         STEPNAME $8   PROGNAME $8  JOBNUMBR $8
         USERID $8;

  Attrib HOSTNAME   Length = $8
                 Label = "Name of remote host"
         MYNAME     Length = $8
                 Label = "Name of user logging on"
         SESSNUM    Length = $5
                 Label = "Remote System Job Id"
         SYSVLONG   Length = $16
                 Label = "SAS System version number";
  SYSVLONG = "&SysVLong";
  SESSNUM  = "&SysJobId";
  HOSTNAME = "&SysSCpl";

  TIOTPTR  = PEEK( PEEK( 540) + 12);
  JOBNAME  = PEEKC( TIOTPTR, 8);
  PROCSTEP = PEEKC( TIOTPTR+8, 8);
  STEPNAME = PEEKC( TIOTPTR+16, 8);

  JSCBPTR  =PEEK( PEEK( 540) + 180);
  PROGNAME =PEEKC( JSCBPTR + 360, 8);

  SSIBPTR  =PEEK( JSCBPTR + 316);
  JOBNUMBR =PEEKC( SSIBPTR + 12, 8);

  JCTPTR   =PEEK( JSCBPTR + 260);
  JOBCLASS =PEEKC( JCTPTR + 47, 1);
  MSGCLASS =PEEKC( JCTPTR + 22, 1);

  ACTPTR   =INPUT( '00'X ||
                 PEEKC( JCTPTR + 56, 3), PIB4.);
  PROGRAMR =PEEKC( ACTPTR + 24, 20);

  SYSTEMID =PEEKC(PEEK(PEEK( 16, 4) + 196, 4) + 16, 4);

  USERID   =PEEKC( PEEK( PEEK( 548) + 108) + 192, 8);

  Label JOBNAME  = 'U157SYSX: Name of executing job'
        JOBNUMBR =
           'U157SYSX: Full Number of executing job'
        USERID   =
           'U157SYSX: userid for owner of job sessio'
        PROGRAMR =
           'U157SYSX: programmer name for tso sessio'
        SYSTEMID =
           'U157SYSX: identifier for running system'
        PROCSTEP = 'U157SYSX: process step name'
        STEPNAME = 'U157SYSX: name of executing step'
        PROGNAME = 'U157SYSX: name of program'
        JOBCLASS = 'U157SYSX: execution class assigned'
        MSGCLASS =
           'U157SYSX: message class for job output';
Run;
```

Here is a dump of the Program Data Vector from this data step.

```
PROGRAMR=DAVID JOHNSON
JOBNAME=FPADHJ
PROCSTEP=IKJACCNT
STEPNAME=SPFA
PROGNAME=IKJEFT01
JOBNUMBR=TSU03390
USERID=FPADHJ
HOSTNAME=MVS
SESSNUM=FPADHJ
SYSVLONG=6.09.0470P042699
JOBCLASS=\
MSGCLASS=Z
SYSTEMID=SYSB
```

Note that the Job and Message classes identify this as an Interactive rather than Batch SAS session. In this case, the session was started by a SAS/Connect session from Windows. We would expect these classes to change, as well as the Job name, the PROC Step and the Step name if the job were running in Batch.

This process is based on a technique shared on SAS/L by Larry Bertolini. It is the realisation of code originally written in Cobol by Gilbert Saint-Flour. His original "COB2JOB" code, and other mainframe techniques are available at http://members.home.net/gsf/tools/

For those unfamiliar with the SAS/L discussion group, further information can be obtained from the archives at http://www.listserv.uga.edu/archives/sas-l.html.

APPENDIX 2: FINDING FILES ON UNIX.

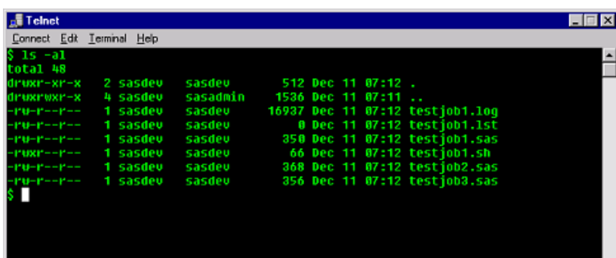
As intimated earlier, the difficulty with the Unix macro is the different data that is available, some of it less specific than in Windows, some of it more so. The ability to define permissions for an owner, a member of the same group, or anyone is certainly an issue Windows has tried to address. Whether it is more successful depends on the opinion and preferred platform of the person arguing the case.

The root aspect of finding a file on Unix is to issue the command as follows.

```
ls -al /export/home/sasprogs*
```

This command will surface a list of all files in the '/export/home' folder with a name beginning with 'sasprogs' and all files found in the folder called 'sasprogs' if it exists. (As a side note here, you can limit this search to the folder called 'sasprogs' if you add another path identifier '/' to the search command. The search request would then look like 'ls –al /export/home/sasprogs/*'. The reason I am not doing that is that the addition of the slash turns a comment on in SAS, and it isn't possible to send this search parameter from the SAS session.)

The parameter group '-al' comprises two parts. The parameter specifies that all files, including hidden files, will be found. The second parameter specifies that a 'long' listing will be provided. This means that file ownership, size and date information is provided. Here then is a sample result from our search.



Let's take each part of the return in turn.

The first part provides 10 bytes to describe the permission set on the found file. The first byte may be one of a number of values, the most common are 'd' which specifies this is a directory, the other is 'l' which indicates this file is a logical link to another file (usually a program).

The next nine bytes comprise three groups of three bytes. The first byte of each group indicates 'r' where the file may be read, the second specifies 'w' where the file may be written to, and the third specifies 'x' where the file may be executed. (Executable files are programs, or batch files, as well as directories.) Where a value is not set, a hyphen identifies the parameter as 'turned off'. The three groups specify the privileges set for the following users; Owner, Owner group, and any user.

Following this we see the file creation date, the file size, the name of the owner and the group to which the owner belonged when the file was created, and finally the file name. Of these, only the creation date needs much explanation. Where the creation date was within the last six months, the date and time are shown, with the year of the date being implied. For older files, only the date is shown, but the year is expressly included.

Our data step code to interpret this data may look like the following.

```
Data ZZZZFILE( Drop = WRITEMON  WRITEDAY
WRITEPRT  TEMPDATE);
    Length FILENAME $50    EXTENSN $8
           ATTCODE         FILESIZE 4
           WRITESD 6       FILEATT $4;
  Infile SYSREQST  TruncOver;
  Input  PERMISS  : $Char10. @;
  If PERMISS Ne: 'total'  Then Do;
    InPut  TEMPNUM  : 4.
           OWNER    : $Char8.
           READGRP  : $Char8.
           FILESIZE : 8.
           WRITEMON : $Char3.
           WRITEDAY : $Char2.
           WRITEPRT : $Char5.
           FILENAME : $Char50.;
    If PERMISS Eq: '-';
    /* If first character is D,
       they are directories */
    If Index( FILENAME, '.') Then Do;
      EXTENSN  = Substr( FILENAME,
           Index( FILENAME, '.') + 1);
      FILENAME = Substr( FILENAME, 1,
           Index( FILENAME, '.') - 1);
    End;
    If Index( WRITEPRT, ':') Then Do;
      TEMPDATE = Input(
          Trim( Left( WRITEDAY) ) ||
          Trim( Left( WRITEMON) ) ||
          Put( DatePart( DateTime() ),
              Year4. ), Date9.);
      If TEMPDATE > Date()  Then
         TEMPDATE = MDY( Month( TEMPDATE),
           Day( TEMPDATE),
           Year( TEMPDATE) - 1);
      WRITESD = Dhms( TEMPDATE,
       Input( Substr( WRITEPRT, 1, 2), 2.),
       Input( Substr( WRITEPRT, 4, 2), 2.),
       0);
    End;
    Else Do;
      TEMPDATE = Input(
          Trim( Left( WRITEDAY) ) ||
          Trim( Left( WRITEMON) ) ||
          Trim( Left( WRITEPRT) ), Date9.);
      WRITESD = Dhms( TEMPDATE, 0, 0, 1);
    End;
    ATTCODE = 0;
    FILEATT = ' ';
    Output;
  End;
  Else Input;
  Format WRITESD  DateTime19.
         FILESIZE Comma10.;
Run;
```

APPENDIX 3: FINDING FILES ON MVS.

Now let's look at the process of finding files on the mainframe. It should be understood by those less familiar with the platform that files are structured differently on a mainframe. Windows and Unix both structure files in hierarchies below the root level of a disk. Windows refers to drives by letter, and also by a virtual location called 'Universal Naming Convention' or 'UNC'. This may take the form of '\\fileserver\directory'. This location may be a section of a physical storage device, and may also be referenced to the operating system by a drive letter. Disk partitions are similarly sections of a physical drive.

Under some new technologies, it may also be a virtual location that is in fact spread across a number of physical devices. Those who have heard the term 'RAID' will recognise the concept of a 'virtual disk'. But whether physical, virtual as in a disk partition, or virtual as in a multiple drive array (RAID): in each case it is

somewhat limited by a physical device. Unix is very similar to this in the sense that a physical storage device is referenced to the operating system by being 'mounted' to the machine. It may similarly, be a whole drive, a section of a drive or a virtual reference to a RAID array.

A Mainframe on the other hand holds a list of files in a system catalogue. This catalogue acts independently of the physical devices on the machine. Indeed accessing a file on a mainframe may cause a tape machine to be called into service to 'restore' a file to a physical device. It's clear that a file that is being restored to a disk, was not present on the disk at the time of the request. The catalogue then does not represent the contents of the physical storage devices. Consequently, searching for files by device does not make much sense.

The machine is not limited by the physical location of the file, and a command like 'Dir C:*.*' isn't really necessary in that context. Of course it can be done. If you want to check a physical device, you can call for a VTOC (Volume Table Of Contents) report. On this platform however, it is irrelevant to anyone who just wants to find files on the system matching a wild card string.

The system catalogue allows the user to search by file name, and the system command for this search will be something like 'ListCat Entry(DAVID)'. In this case, all files with a 'high level qualifier' of 'DAVID' will be returned. It is very like searching under Windows with the command 'Dir DAVID*'. The difference is that the search returns every file matching the wild card, whether or not it is currently on the machine.

Sure, Dos let you search all sub directories below a given root for files matching a wild card, and Windows allows it across multiple drives through its explorer. (Actually, Windows cheats a little, because it performs a recursive search, starting at the root of each drive specified, and then searching each sub directory in turn. It literally 'walks' across the directory tree.)

Unix is also a lot friendlier in allowing machine wide searches using the 'grep' functionality, but you need to be fairly sophisticated in the way you issue commands to the operating system. In any case, a file on a CDRom will not be found, if the CDRom is not 'mounted' on the machine. To belabour the point, OS/390 can call for a device to be mounted to return the file to the machine.

So what we have established is that OS/390 will allow much wider searches than those we may be used to in Windows or Unix. So our command 'ListCat Entry(DAVID)' can be used to find files in a very wide area, and quite quickly. (Remember that we are reading a relatively small System catalogue, not 'walking' across a tree structure of directories.) Unfortunately, speedy though the command is, it is not very informative. Here is the abbreviated result of a search using the high level qualifier ……

```
VALUE=1IDCAMS   SYSTEM SERVICES   TIME: 15:45:40
09/26/01   PAGE   1
VALUE=0NONVSAM ------- DAVID.JCL.PDS
VALUE=IN-CAT --- CATALOG.DEV1CAT
VALUE=0NONVSAM ------- DAVID.PROFILE
VALUE=IN-CAT --- CATALOG.DEV1CAT
VALUE=0NONVSAM ------- DAVID.SAS.MACROS
VALUE=IN-CAT --- CATALOG.DEV1CAT
VALUE=0NONVSAM ------- DAVID.SAS.PDS
VALUE=IN-CAT --- CATALOG.DEV1CAT
VALUE=0NONVSAM ------- DAVID.SASSRCE
VALUE=IN-CAT --- CATALOG.DEV1CAT
VALUE=0NONVSAM ------- DAVID.SASUSER
VALUE=IN-CAT --- CATALOG.DEV1CAT
VALUE=0NONVSAM ------- DAVID.SAS.DATA
VALUE=IN-CAT --- CATALOG.DEV1CAT
VALUE=1IDCAMS   SYSTEM SERVICES
TIME: 15:45:40   09/26/01   PAGE   2
VALUE=0          THE NUMBER OF ENTRIES PROCESSED WAS:
VALUE=NONVSAM ---------------7
VALUE=TOTAL -----------------7
VALUE=0   THE NUMBER OF PROTECTED ENTRIES SUPPRESSED WAS
0
VALUE=0IDC0001I FUNCTION COMPLETED, HIGHEST CONDITION
CODE WAS 0
```

All we have established is that there are 7 files matching the given high-level qualifier in the system catalogue. We don't know whether we are looking at a set of SAS programs, a Library of SAS data sets and catalogues, or a flat text file. To do this, we need to get further information. Before we do that though, lets move what we have already discovered from the system interface into the SAS session.

Now on a mainframe, you can't just issue Call System or X commands in the same way. But OS/390 has a wonderful feature that allows you to call any OS/390 process or 'PROC' by calling it as if it were a SAS procedure. In this way, a very useful mainframe PROC called IDCAMS can be called from within a SAS session. It looks like a SAS Procedure, but in fact it is a system PROC, as the log will tell us.

(Note that on OS/390, a system process or program is called a PROC. Throughout this document, the term 'PROC' will refer to a process on the OS/390 platform. It should not be confused with a SAS Procedure, which will be referred to as a Procedure.)

To call the PROC, you need to provide a command input, which is called a system input or SYSIN file. So we define a SYSIN 'DSN' to our session with a filename statement. (The term DSN refers to the name of a file created on the OS/390 platform. A DSN is not a SAS Data Set. The term 'DSN' will refer exclusively to a file on the OS/390 Operating System. A Data Set on the SAS System will be referred to as a SAS Data Set.) The file name parameter we pass it causes the operating system to create a new temporary file.

```
FileName SYSIN  '&temp1' Space = (Trk, (1.1) );
```

Then we provide a file to receive the results of the PROC, which we can then handle in essentially the same way we handled the piped file we explored when we were looking at Unix commands. This file is larger (by 16 times) than the command file to ensure there is space to hold the search results.

```
FileName IDOUT '&temp2' Space = (Cyl, (1.1) )
   RecFm = Vba   LRecL=255;
```

Then we build a command string, and save it in the SYSIN file, using very simple and common SAS statements. The command we are sending is asking the Operating System to list from the catalogue, all entries that start with a given High Level Qualifier. We passed the HLQ within the macro parameter MDIR. Note that we have used the same parameter name and function with the OS/390 call that we used on Unix and Windows. This will allow us to build a macro that can be called by code we developed on another type of platform.

The command also instructs that the result of the command will be passed to an output DSN. We specify that the DSN is catalogued as 'IDOUT', which is the DSN we created above.

```
Data _NULL_;
  File SYSIN;
  STRING = " LISTCAT LEVEL('" ||
    Trim( "&MDIR") || "') OUTFILE('IDOUT')";
  Put STRING $Char80.;
Run;
```

Finally, we call the PROC with what looks like a call to the IDCAMS Procedure. Note the entry in this extract from the SAS log that reminds us that IDCAMS is not a SAS procedure, despite our having apparently issued a SAS statement.

```
1625    Proc IdCams;
NOTE: An external program, not a SAS procedure, is
being executed.

1626    Run;

NOTE: The PROCEDURE IDCAMS used the following
resources:
      Task  memory - 490K (20K data, 470K program)
      Total memory - 16052K (12032K data, 4020K
program)
```

The results are in our output file, as a consequence of our using the statement "OUTFILE('IDOUT')" in the SYSIN file. We can read and parse this file, and produce a set of results.

```
** Open and parse the results of the find process;
** ZTMPFILE is the abbreviated list of files found;
** We will add this with further details to ZZZZFILE later;
** ZSUMMARY captures the total count, the number of
** hidden files and summarises the file types
** identified;

Data ZTMPFILE( Drop = VALUE DATETIME SUMMARY FILENUM)
     ZSUMMARY( Label = 'Summary list of DSNs found'
               Drop = VALUE SUMMARY FILENAME CATALOG);

  Length FILETYPE $8  FILENAME $80;
  Retain FILENAME FILETYPE ' ' SUMMARY 0 DATETIME;

  InFile IDOUT TruncOver;
  Input @1 VALUE $200.;

** The catalog name is written AFTER each entry;
  If VALUE Eq: 'IN-CAT'  Then  Do;
    CATALOG = Trim( Substr( VALUE,
                        Index( VALUE,
                               ' CATALOG.') + 9) );
    Output ZTMPFILE;
  End;

  Else If Index( VALUE, '-------')  And
          Not SUMMARY   Then Do;
    FILETYPE = Trim( Substr( VALUE, 2,
                        Index( VALUE, ' --')-1) );
    FILENAME = Trim( Substr( VALUE,
                        Index( VALUE, '- ')+1) );
  End;

  Else If Index( VALUE, 'SYSTEM SERVICES')   Then  Do;
    VALUE = CompBl( Substr( VALUE,
                        Index( VALUE, 'TIME:') ) );
    DATETIME = DHms( Input( Substr( VALUE, 16, 8),
                          Mmddyy8.),
                    0, 0,
                    Input( Substr( VALUE, 7, 8),
                        Time8.) );
    If DateTime() - DATETIME > 30  Then
        Put 'ERROR: This data is out of date.';
  End;

 /* Start after <entries processed> line */
  If SUMMARY  Then Do;

    If Index( VALUE, 'PROTECTED ENTRIES')  Then Do;
      FILETYPE = 'HIDDEN';
      FILENUM = Input( Scan( Reverse( VALUE), 1 ), 8.);
    End;

    Else If Index( VALUE, 'HIGHEST CONDITION')
         Then Do;
      FILETYPE = 'CONDCODE';
      FILENUM = Input( Scan( Reverse( VALUE), 1 ), 8.);
    End;

    Else Do;
      VALUE = Compress( VALUE, '-');
      FILETYPE = Substr( VALUE, 1,
                         Index( Value, ' ') - 1 );
      FILENUM  = Input( Substr( VALUE,
                           Index( Value, ' ')+1),
                       8.);
    End;

    Output ZSUMMARY;

  End;

  If Index( VALUE, 'ENTRIES PROCESSED') Then
     SUMMARY = 1;
  Format DATETIME DateTime19.1;

Run;
```

Here is an extract from the Program Data Vector of the ZTMPFILE Data Set, to illustrate that we have detected all the OS/390 DSNs that we listed in the file dump above:

```
FILETYPE=NONVSAM FILENAME=DAVID.JCL.PDS CATALOG=DEV1CAT
FILETYPE=NONVSAM FILENAME=DAVID.PROFILE CATALOG=DEV1CAT
FILETYPE=NONVSAM FILENAME=DAVID.SAS.MACROS
CATALOG=DEV1CAT
FILETYPE=NONVSAM FILENAME=DAVID.SAS.PDS CATALOG=DEV1CAT
FILETYPE=NONVSAM FILENAME=DAVID.SASSRCE CATALOG=DEV1CAT
FILETYPE=NONVSAM FILENAME=DAVID.SASUSER CATALOG=DEV1CAT
FILETYPE=NONVSAM FILENAME=DAVID.SAS.DATA
CATALOG=DEV1CAT
```

Now here is a dump of the summary we captured in the ZSUMMARY Data Set. This summary was captured because it provides diagnostic information we can use in the event of our search returning unexpected results.

```
FILETYPE=NONVSAM DATETIME=26SEP01:15:47:43.0 FILENUM=7
FILETYPE=TOTAL DATETIME=26SEP01:15:47:43.0 FILENUM=7
FILETYPE=HIDDEN DATETIME=26SEP01:15:47:43.0 FILENUM=0
FILETYPE=CONDCODE DATETIME=26SEP01:15:47:43.0 FILENUM=0
```

Note that we have a summary of the file types that the ListCat command would return. We have also captured the total number of files, the 'hidden' file count and the condition code for future reference.

This is in fact our only direct call to the OS/390 Operating System. We will interface with the Operating System again shortly, but we will use a SAS procedure that is only available for the OS/390 platform.

For now, we are going to take this information and surface more information on the files we located. We do this by reading the system information on the file, called the 'Job File Control Block' or JFCB. There is a very good paper on this subject published by Don Stanley of Sysware consulting in New Zealand. His paper details are at the end. Don's work, and a little empirical analysis on other parts of the JFCB, indicates that we can find details on the file type, the file creation date, the file size, and indeed most of the things we wanted to know about files on Windows.

The following data step uses an InFile parameter called JFCB to surface details of the file we are opening. We can read and parse the JFCB string, and save information on the file. We use a simple data step to read the parameters of the filename we passed in the 'Mdir' macro symbol to our InFile statement.

```
Data ZZZZJFCB( Drop = BYTE81 BYTE82);

  Attrib FILENAME  Length = $54
                   Label = 'Data set name from JFCB'
```

```
          EXTENSN   Length = $8
                    Label = 'Extension for file name'
          CREATESD  Length = 8      Format = Date9.
                    Label = 'Date time file created'
          ACCESSSD  Length = 8      Format = Date9.
                    Label = 'Date time file accessed'
          WRITESD   Length = 8      Format = DateTime19.
                    Label = 'Date time file updated'
          FILEATT   Length = $8
                    Label = 'File type attributes'
          FILESIZE  Length = 8      Format = Comma14.
                    Label = 'File size in bytes'
          LASTOWNR  Length = $8
                    Label = 'Id of user who last changed
PDS Member';

/* These were provided for compatibility with Windows NT */
/* The WRITESD will be retrieved for PDS members only,  */
/*  for other DSNs the date will be missing.           */
/* The ACCESSSD date will always be missing.  It will be */
/*  updated with the creation date of a PDS member.     */
/* These are set to missing to avoid uninitialised messages      */
  ACCESSSD = .;
  WRITESD  = .;
  FILESIZE = .;
  LASTOWNR = ' ';

  InFile "&MDsn"  UnBuffered JFCB = JFCB  RecFm = U;

  /* Assign the DSName to the filename,
   and the PDS member name to the extension */
  DSNAME    = Substr( JFCB, 1, 44);
  EXTENSN   = Substr( JFCB, 45, 8);

  /* We use the Byte(0) function to delete blanks to ensure cross
   platform compatibility between ASCII and EBCDIC encoding.
   This macro may be migrated between Windows & OS/390 platforms
*/
  If Compress( EXTENSN, Byte(0) ) || '.' Ne: '.'  Then
    FILENAME = Compress( DSNAME || '(' ||
               EXTENSN || ')' );
  Else  FILENAME = DSNAME;

  /* Is the data set catalogued? */
  If Substr( JFCB, 53, 1) = '1.......'b  Then
          CATLG = 'C';
  Else  CATLG = 'U';

  /* Get the DSN creation date */
  BYTE81    = Input( Substr( JFCB, 81, 1), Ib1.);
  BYTE82    = Input( Substr( JFCB, 82, 2), Ib2.);
  CREATESD  = Input( Put( BYTE81 + 1900, 4.) ||
                  Put( BYTE82, z3.), Julian7.);

 /* If the data set is part of a Generation Data Group, this
    flag is set.  */
  If Substr( JFCB, 87, 1) = '......1.'b  Then Do;
    GDG = 'Y';
    Substr( FILEATT, 3, 1) = 'G';
  End;
  Else GDG = ' ';

  /* Identify Temporarily or Permanently catalogued DSNs */
  If Substr( JFCB, 88, 1) = '.......1'b  Then
       KEEPSTAT = 'T';
  Else KEEPSTAT = 'P';

 /* The Data Set disposition. If this is SHR (share) we will
    create an attribute of (R) for read only. */
  If       Substr( JFCB, 88, 1) = '11......'b  Then
      DISP = 'NEW';
  Else If Substr( JFCB, 88, 1) = '10......'b  Then
      DISP = 'MOD';
  Else If Substr( JFCB, 88, 1) = '01......'b  Then
      DISP = 'OLD';
  Else If Substr( JFCB, 88, 1) = '01..1...'b  Then
      DISP = 'SHR';
  If DISP Eq 'SHR'  Then  Substr( FILEATT, 2, 1) = 'R';

  /* The data set organisation,
    note the partitioned sequential (PS) type  */
  If       Substr( JFCB, 99, 1) = '1.......'b  Then
         DSORG = 'IS';
  Else If Substr( JFCB, 99, 1) = '.1......'b  Then
         DSORG = 'PS';
  Else If Substr( JFCB, 99, 1) = '..1.....'b  Then
         DSORG = 'DA';
  Else If Substr( JFCB, 99, 1) = '......1.'b  Then
         DSORG = 'PO';
  Else   DSORG = 'OT';
    Substr( FILEATT, 4, 2) = DSORG;

  /* The file record layout.
    Fixed Blocked (FB) records in a PS organisation denote
     a partitioned data set.
    You will use Proc Source to get
     member owner, size and date information */
  If       Substr( JFCB, 101, 1) = '11......'b  Then
       RECFM = 'U  ';
  Else If Substr( JFCB, 101, 1) = '10.0.00.'b  Then
       RECFM = 'F  ';
  Else If Substr( JFCB, 101, 1) = '01.0.00.'b  Then
       RECFM = 'V  ';
  Else If Substr( JFCB, 101, 1) = '10.1.00.'b  Then
       RECFM = 'FB ';
  Else If Substr( JFCB, 101, 1) = '01.1.00.'b  Then
       RECFM = 'VB ';
  Else If Substr( JFCB, 101, 1) = '10.1.10.'b  Then
       RECFM = 'FBA';
  Else If Substr( JFCB, 101, 1) = '01.1.10.'b  Then
       RECFM = 'VBA';
  Else If Substr( JFCB, 101, 1) = '10.1.01.'b  Then
       RECFM = 'FBM';
  Else If Substr( JFCB, 101, 1) = '01.1.01.'b  Then
       RECFM = 'VBM';
  Substr( FILEATT, 6, 3) = RECFM;

  /* Standard DCB sizing information */
  BLKSIZE = Input( Substr( JFCB, 102, 3), Ib3.);
  LRECL   = Input( Substr( JFCB, 105, 2), Ib2.);

  /* A DSN may span up to five volumes. Trap all volumes.  Flag Multi
volume DSNs to the file attribute string. */
  VOLSER1 = Substr( JFCB, 119, 6);
  VOLSER2 = Substr( JFCB, 125, 6);
  VOLSER3 = Substr( JFCB, 131, 6);
  VOLSER4 = Substr( JFCB, 137, 6);
  VOLSER5 = Substr( JFCB, 143, 6);
  If Length( Compress( VOLSER1, Byte(0) ) ) <
     Length( Compress( VOLSER1 || VOLSER2 || VOLSER3 ||
                       VOLSER4 || VOLSER5, Byte(0) ) )
       Then Substr( FILEATT, 1, 1) = 'M';

Run;
```

Note the types of files that we identify from the file Record Format, or RECFM. The record format 'PSF' refers to a Partitioned Sequential Fixed structure. This is the structure we encounter with a library of SAS data sets. Here is the relevant data from the Program Data Vector on a SAS data library.

```
FILENAME=DAVID.SAS.DATA
EXTENSN=
CREATESD=05JUN2001
ACCESSSD=
WRITESD=
FILEATT=PSF
FILESIZE=
LASTOWNR=
LRECL=27640
BLKSIZE=27640
VOLSER1=DAVIDS
```

```
VOLSER2=
VOLSER3=
VOLSER4=
VOLSER5=
```

The BlkSize and LRecL parameters refer to the physical structure of the file, the LRecL (Logical Record Length) is the length of a record, and the BlkSize refers to the size of the data block that will contain the logical records we are creating. These are often equal.

The other types we commonly see are 'PSFB' and 'POFB'.

PSFB is Partitioned Sequential Fixed Blocked. This is a file structure you find with normal text files. The OS/390 DSN is one large text file comprising records of an equal length (i.e. Fixed). These are written into blocks (i.e. Blocked). The blocks are stored one after the other (i.e. Sequential) and are a circumscribed piece of data (i.e. Partitioned).

On the other hand, we can have a similar type of file that stores a series of individual text members. The difference with this file is that it also holds a directory of pointers to the beginning and end of blocks of data that form files in their own right.

The structure is PO for Partitioned Ordered, and refers to a series of ordered sections within a much larger file structure. This structure is commonly called a Partitioned Data Set or PDS, and is the place most programmers store programs, blocks of code and 'Card Libs'.

(Card Libs is a reference to the time when programmers passed instructions to computers on decks of cards. When their storage was moved onto tape, and later on disk the reference to card libraries remained. For the same archaic reason, programs are often called 'decks'.)

Here is the output produced for a PDS.

```
FILENAME=DAVID.SAS.PDS
EXTENSN=
CREATESD=05JUN2001
ACCESSSD=
WRITESD=
FILEATT=POFB
FILESIZE=
LASTOWNR=
LRECL=80
BLKSIZE=27680
VOLSER1=DAVIDS
VOLSER2=
VOLSER3=
VOLSER4=
VOLSER5=
```

FURTHER DETAILS ON FILES FOUND

To get a list of the programs (members) of a PDS, we need to employ a Procedure in SAS written for the OS/390 platform. It is called the SOURCE Procedure and allows us to access individual members of a PDS. When we give it the parameter DIRDD, the Procedure returns the Directory Data Descriptors to the output destination we specify.

```
** Create a temporary DSN to hold the search results;
** The Blocksize is defined for 3390 device types ;
** This setting will allow for analysis of :
    (27680 / 80) * 15     members. That is 5190.   ;
FileName TmpDirDD '&Temp' Disp = ( New, Delete, Delete)
        Space = ( Trk,( 1, 1) )  DSOrg = PS
        RecFm = FB  LRecl = 80  BlkSize = 27680;

** Surface the list of members to the file,
    suppressing printing to the log;
  Proc Source  InDD   = "&MDsn"
               DirDD = TmpDirDD  NoData  NoPrint;
  Run;


** Reading and validation of the return from the search;
Data ZMEMBATT( Drop = IND  VER   MOD     DATEBLOK  YEAR
                      DAY  HOUR  MINUTE  TTR);
  Attrib MEMBER    Length = $8
              Label = 'PDS member name'
         CREATESD  Length = 8      Format = Date9.
              Label = 'Date time file created'
         WRITESD   Length = 8      Format = DateTime19.
              Label = 'Date time file updated'
         SIZE      Length = 8      Format = Comma14.
              Label = 'Number of lines in PDS member'
         INITSIZE  Length = 8      Format = Comma14.
              Label =
                'Initial number of lines in PDS member'
         MODLINES  Length = 8      Format = Comma14.
              Label =
                'No. of lines last mod in PDS member'
         LASTOWNR  Length = $8
              Label =
                'Id of user last changed PDS Member'
         VERSION   Length = $6
              Label = 'Version number of PDS member';
  LASTOWNR = 'NOSTATS';

  InFile TmpDirDD  TruncOver;
  Input @ 1 MEMBER   $8.
        @ 9 TTR      PIb3.
        @12 IND      PIb1. @;

/* Member has stats */
  If 2 * Mod( IND, 32) = 30  Then Do;
    Input @13 VER      PIb1.
          @14 MOD      PIb1.
          @17 DATEBLOK $Char10.
          @27 SIZE     PIb2.
          @29 INITSIZE PIb2.
          @31 MODLINES PIb2.
          @33 LASTOWNR $8.;

    VERSION = Put( VER, 3.) || '.' || Put( MOD, Z2.);

    YEAR     = 1900 + (Substr( DATEBLOK, 1, 1) = '01'x)
                    * 100 +
               Input( Substr( DATEBLOK, 2, 1), Pk1.);
    DAY      = Input( Put( Substr( DATEBLOK, 3, 2),
                           Hex3.), 8.);
    CREATESD = Input( Put( YEAR, Z4.) ||
                      Put( DAY, Z3.), Julian7.);

    YEAR     = 1900 + (Substr( DATEBLOK, 5, 1) = '01'x)
                    * 100 +
               Input( Substr( DATEBLOK, 6, 1), Pk1.);
    DAY      = Input( Put( Substr( DATEBLOK, 7, 2),
                           Hex3.), 8.);
    HOUR     = Input( Substr( DATEBLOK, 9, 1), Pk1.);
    MINUTE   = Input( Substr( DATEBLOK, 10, 1), Pk1.);

    WRITESD  = Dhms( Input( Put( YEAR, Z4.) ||
                            Put( DAY, Z3.), Julian7.),
                     HOUR, MINUTE, 0);
  End;

  Else Input;
  Format CREATESD Date9. WRITESD DateTime19.;
Run;
```

Using this data step, here is the data we can surface on the PDS.

```
MEMBER=CONNECT
```

```
CREATESD=
WRITESD=
SIZE=
INITSIZE=
MODLINES=
LASTOWNR=NOSTATS
VERSION=
YEAR=
DAY=
HOUR=
MINUTE=

MEMBER=TSOSEC
CREATESD=07JUN2001
WRITESD=07JUN2001:14:54:00
SIZE=289
INITSIZE=289
MODLINES=0
LASTOWNR=DAVID
VERSION=1.00
YEAR=2001
DAY=158
HOUR=14
MINUTE=54
```

Note that our first member has missing values for most of the statistics, and the Last Owner attribute has been forced to 'NOSTATS' by our code to indicate that no statistics have been saved. This particular member was written to the SAS PDS library with a SAS Connect session and the Upload Procedure. In this process, the system statistics are not updated.

The second member has been updated in a TSO session on OS/390, and we can see that it has not been modified since it was written. The update on TSO, with the 'STATS' option turned on for the library, has ensured that we now have stats saved for the PDS member.

The only other aspect of mainframe libraries we haven't explored is the SAS library. Windows and Unix allow us to identify SAS data sets in our file find routines by the file extension (.SD2 in Windows Version 6, .SSD01 in Unix Version 6, .SAS7BDAT in Version 8). On the OS/390 environment, the whole library of SAS data sets, (catalogues, indexes and so on) is stored as one DSN in the OS/390 system catalogue.

However, the SAS System provides us with the solution for reading the contents of the library. Consider the DSN we found called 'DAVID.SAS.DATA'. Here are the SAS members of that library, which we read after assigning that DSN to our session with the library reference 'SUGI'.

```
25   Proc Sql  NoPrint  STimer  _Method;
NOTE: The SQL Statement used 0.01 seconds.

26
27    Create Table DATASETS As
28      Select MEMNAME, NOBS, NVAR, CRDATE, MODATE
29      From Dictionary.Tables
30      Where LIBNAME = 'SUGI';

NOTE: SQL execution methods chosen are:

      sqxcrta
          sqxfil
              sqxsob

NOTE: Table WORK.DATASETS created, with 118 rows and 5
columns.

NOTE: The SQL Statement used 0.74 seconds.

31
32   Quit;
NOTE: The PROCEDURE SQL used 0.01 seconds.
```

Here is a test of the data that we have captured with the above process.

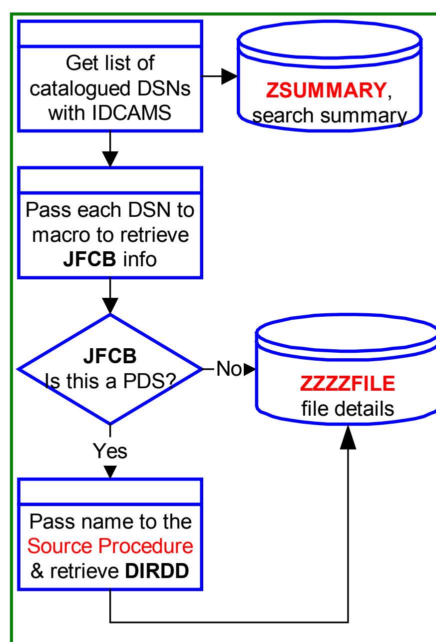
```
34   Data _NULL_;
35     Set DATASETS;
36     If _N_ < 4  Then Put MEMNAME= CRDATE= MODATE=;
37   Run;

MEMNAME=A CRDATE=02APR01:13:44:56
MODATE=02APR01:13:44:56
MEMNAME=AAAAOPTN CRDATE=29MAR01:17:43:09
MODATE=29MAR01:17:43:09
MEMNAME=ACCESLOG CRDATE=16MAR01:08:49:25
MODATE=11APR01:16:21:14
NOTE: The DATA statement used 0.28 seconds.
```

Naturally, you don't have to use the SQL view approach; the DataSets, Contents and Catalog Procedures all provide means for finding the contents of a SAS library. This is probably the easiest part of the macro to test, since it runs on any platform where SAS is installed.

Putting all of this together into a macro can seem a little daunting, but perhaps the following diagram will make it easier to follow. We have treated each part of the process as a separate file process, or object. That allows us substantial flexibility in calling processes as we need them, and also allows us to customise the results for certain file types very easily. The three parts are:

- Use Proc IDCAMS to get the initial file list
- Read the JFCB on each file in turn
- Read the DIRDD on PDS files.



One very important warning on the above… The process involves reading the system catalogue initially. This returns information that will not affect the status of a DSN. However, when we read each DSN to retrieve the JFCB information, the DSN needs to be available to the Operating System to be read. If it was a tape file, or is an old file that the system has 'archived', then your request will cause the storage tape to be mounted and the file restored to disk. If you use a very common High Level Qualifier in your search process you will alienate

- your Operators with the number of cartridge mounts,
- your fellow users with your hogging a limited resource,
- your Storage Management people with the restoration of so much 'old data'

and make yourself unhappy with the time it will take to retrieve and read a lot of cartridges. So be careful with the search parameter.

The original paper on JFCB and DIRDD are available from Don Stanley's website at http://www.geocities.com/don_stanley_nz/sas_tips/jfcb.htm.

CONCLUSION

When we review the development process we undertook for our 'cross platform macros', the following guidelines emerge:

- Devise any new macro as generically as possible, it may become necessary on another platform
- Research the macro call carefully. If you are parsing the return from a system call, then look at what else may be available and either add that to the macro return, or document it in the macro for future use.
- Set up a standard naming convention for system and generic macros and their parameters, and never mix these with the names used for application macros.
- Set aside a block of data set names, library and filename references. Use these always and only with your cross platform macros.
- Document the macros thoroughly and carefully. In a year from now, you may be the person trying to understand the documentation to fix a problem, or extend functionality.

TRADEMARK CITATION

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

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

REFERENCES

SAS Institute Inc (1995), *TS460: Accessing External DLLs with SAS 6.1x for Win32s*, Cary, NC, SAS Institute Inc.

Barron, David L & Hemedinger, Chris (1996), "Accessing Dynamic Link Library Routines with the SAS System for Windows", *Observations: The Technical Journal for SAS Software Users*, First Quarter 1996.

SAS Institute Inc (1996), *Microsoft Windows Environment: Changes and Enhancements to the SAS System, Release 6.11*, Cary, NC, SAS Institute Inc.

Johnson, David H (1997), "DLLs, APIs and SAS", SAS Melbourne User Group.

Johnson, David H (2000), "Have SAS, will travel", SAS European User Group International, Dublin, SAS Institute Inc.

ACKNOWLEDGEMENTS

My clients who have asked for the robustness and functionality that required this development to take place.

The SUGI committee and especially the Advanced Tutorials section chair, Deb Cassidy, for supporting the production of this paper.

The SAS Institute Inc for their documentation, and the 'Observations' publications which provide a forum for sharing ideas and experiences.

The helpful people who post ideas and assistance to SAS/L, the global SAS discussion list you can access at http://www.listserv.uga.edu.

Last but not least, my family for their support and patience.

CONTACT INFORMATION

Your comments, suggestions and questions are valued and encouraged. Please contact the author:

David Johnson
DKV-J Consultancies
C/- 'Bonds Cottage,
Holmeswood Rd
Holmeswood nr Rufford
Lancashire England L40 1UA
Business Phone: +44 (0)7005 98 0828
Fax: +44 (0)7092 25 9556
Email: sugi28@dkvj.com
Web: http://www.dkvj.com

DKV·J Consultancies
Business Information Systems

© 2000-2003, drawn with SAS/Graph®