Paper 285-25

Using SAS® Under OpenVMS® at First Health

Jack Hamilton, First Health, West Sacramento, California USA

ABSTRACT

This paper discusses how SAS Software is used by the Metrics Department of First Health. Some basic OpenVMS® concepts are discussed. This paper is for aimed to beginning and intermediate level users of SAS and OpenVMS.

UPDATES

I did not have access to Version 8 when I wrote this paper. After V8 becomes available, I will revise this paper as needed. I will also incorporate comments received at SUGI. The new version will be available on my web page at the address shown below. You will be able to register for notification of future updates.

INTRODUCTION

The OpenVMS operating system is licensed from Compaq Corporation, which owns it by virtue of having purchased Digital Equipment Corporation, the original developer. It runs on COMPAQ Alpha and VAX hardware. It has many similarities to other mainstream operating systems. A few of the basic concepts are explained below. Keep in mind that all of these explanations are simplified, perhaps even oversimplified.

COMMANDS

OpenVMS commands can refer to executable programs or to command procedures written in DCL. The various methods OpenVMS uses to define and resolve names are too complicated to go into in detail here. A typical command consists of a command name followed by optional parameters and qualifiers. The command to run a short SAS program might be:

```
$ sas /errabend myprogram.sas
```

The `$` is a convention used to represent the OpenVMS prompt. `SAS` is the name of the program to execute. `/errabend` is a qualifier, and for the SAS command must be a SAS system option (because of an OpenVMS parsing restriction, some parameters must be specified using a different name). `myprogram.sas` is a parameter, and for the SAS command is the name of the program to run.

DCL

DCL (DIGITAL Command Language) is the OpenVMS shell and scripting language. It's similar to the DOS command language in its lack of support for structured programming, but it does give access to system information through built-in functions.

DCL commands can be combined into command procedures, similar to DOS batch files. A DCL command procedure (usually called a com file) is executed by preceding its name with an @ sign:

```
$ @ mt$login   ! Run common login
```

The comment character in DCL is the exclamation mark (!).

DRIVES AND DIRECTORIES

The OpenVMS file structure is similar to that of DOS and Unix systems except for how directories are specified. A complete file name consists of device (or drive) name, a directory name (which may have many levels), a file name, and a file extension. All parts of names are uppercase, but names may be longer than 8 characters. An example of an OpenVMS file name (my login file, in this case) is `asreshome:[hamiltja]login.com`.

LOGICAL NAMES

An OpenVMS logical name (usually just called a logical) provides a way to refer to one or more drives, directories, or files. A logical used in this way is similar to an MVS DDName, but is more flexible. In general, an appropriately defined logical name can be used anywhere you might use a a drive name, directory name, or file name.

We use logical names for two primary purposes: to eliminate hardcoded file locations in programs, and to specify multiple locations which will be searched for a file. In addition, many software packages, including SAS, look for various logicals to control execution. All of these uses will be explained in greater detail below.

Logicals are established with the DEFINE command. Here are some examples of logical definitions:

```
$ define sas$news MT$COMSAS:SASNEWS.TXT
$ define HIAA-INIT HIAA-PGMS:HIAA-INIT.SAS
$ define  dev-core-xtrn-data   -
                    dev-core-xtrn-data-w,
                    stg-core-xtrn-data
$ define  stg-core-xtrn-data   -
                    stg-core-xtrn-data-w,
                    prd-core-xtrn-data
$ define  prd-core-xtrn-data
                    prd-core-xtrn-data-w
```

Logicals are widely used in OpenVMS; my login session usually has more than a thousand logicals, most of them defined by the system.

Note that although logicals usually refer to file system objects, a logical can be any text string. Logicals are occasionally used to set program options or pass values.

The values of logicals are available to SAS programs through the GETLOG function, but none of our programs need to use that function, as the logicals are translated into complete file system names internally by the SAS supervisor.

SYMBOLS

Symbols are similar to environment variables in DOS or Unix. Local symbols are used as variables to store values in DCL command procedures. Global symbols are used to store values during a session, or to pass values between command procedures. Symbols can also be used to define aliases for command procedures, allowing them to be used as commands.

Here are some examples of symbol definitions:

```
$ PROJECTS == "@mt$comutil:projects.com"
$ SAY == "write sys$output"
$ MY_TERMTYPE == "vt4xx"
```

My login session has approximately 200 symbols defined.

DCL symbols are available to SAS programs through the GETSYM and %SYSGET functions

EQUIPMENT OVERVIEW

First Health has seven machines running OpenVMS. They total approximate 13 gigabytes of memory and 3 terabytes of disk space. The primary SAS machine has 2 gigabytes of memory and 986 gigabytes of disk storage.

COMMON LOGIN PROCEDURE

OpenVMS users have a login file (a standard DCL command procedure) which is automatically executed upon logging in to the system.

My department, Metrics, was created from the merger of several smaller departments over the last few years. Because there was no system-wide standard login, new employees usually just copied their login file whom whoever happened to be training then in OpenVMS. The new employees would then make their own changes. As a result, there were many different login procedures, and OpenVMS wasn't set up exactly the same way for any two colleagues. It was not uncommon to redefine system commands, so even standard com files would fail for some people. We couldn't easily explain the steps needed to edit a file, because everyone had tailored the editor to use an idiosyncratic set of keys In short, it was a mess.

Persuading everyone to use a common login file took some doing; it fact, the process is not yet complete. Users who have been around for a while become quite attached to their setups, even if they're hard to use. As the saying goes, change is hard, even from bad to good.

We got around that in two ways: by making new applications available only to users of the new common login (which encourages them to upgrade); and by setting up user exits so users could still do some personalization.

WHAT THE COMMON LOGIN DOES

The current common login performs 6 functions:

- It sets up symbols to define commands and options.
- t defines logicals for department projects.
- It defines logicals for department production data.
- It customizes how SAS will run.
- It customizes terminal setup.
- It calls a user exit.

USER EXITS

We could, of course, have issued an edict saying that everyone must use the common login that we provide, with no changes, and we could have come in at night and erased any non-compliant files. But we didn't.

Our department philosophy is that there's usually more than one way to do things. Some ways are better than others, and should be encouraged. But even the "bad" ways aren't (usually) totally bad. There's no need to create resentment by forcing people. Almost everyone has eventually come around, just because they see how much easier the new way is to use.

User exits provide a way to allow everyone some personalization while still having the same general setup and capabilities as everyone else.

COMMON SYMBOLS

The common login sets up approximately 100 symbols, in 4 different categories:

COMMAND DEFINITIONS

Approximately 40 command definitions are created. Here are two of them as an example:

```
$ u*tils   ==  "@mt$comutil:utils.com"
$ dirs*ize  ==  "dir ''my_diropts' /size"
```

The first line assigns the command UTILS to call a utility com file (described below). Note the use of logical, mt$comutil, instead of a hardcoded drive and directory name. The * tells OpenVMS that the command can be abbreviated to its first letter.

The line command assigns the command DIRSIZE to call the built-in command DIR with the /SIZE option. MY_DIROPTS is a symbol that a user can set in their user exit login file. I have it defined to mean `/width=(filename=40,size=12)`, so when I type DIRS the executed command is

```
dir /width=(filename=40,size=12) /size
```

This is more suitable for long filenames and a 132-character screen. MY_DIROPTS is used in several other directory symbols, and is an example of what can be done with user exits.

OPTION VALUES

A dozen symbols are used to store the values of various settings. MY_DIROPTS is one. Another important one, which will be discussed below, is MY_DATALEVEL. A few user-information settings, such as the user's home drive, are also placed into symbols for easier use in com files.

DATABASE LOCATIONS

Our corporate data are stored, for the most part, in INGRES® databases running on other OpenVMS machines. We read these databases using SAS/ACCESS. For operation reasons, the databases are occasionally moved from machine to machine. When this happened in the past, we had to locate which referred to the old location, and change it to use the new location.

We have solved that problem by storing the database locations in symbols. The symbols are created in one com file, which is the only thing that has to be changed when databases are moved. A location symbol is defined like this:

```
$ mt_rpt_zipcode   == "mars::zipcode"
```

The symbol is used in PROC SQL like this:

```
connect to ingres as provider
     (database="%SYSGET(MT_RPT_PROVIDER)");
```

TERMINAL CONTROL CHARACTERS

DCL com files can interact with the user, displaying or prompting for information. It's helpful to add a limited amount of highlighting for emphasis. The common login queries the terminal (always a terminal emulation program here) and sets the values of value DCL symbols to the values that must be sent to the terminal to produce a particular effect. Doing this in the common login means that each com file doesn't have to figure it out for itself.

PROJECT LOGICALS

Most of our work is done in projects. Because the Metrics department is primarily an analysis and ad hoc reporting group, projects are usually fairly small, involving no more than 3 people over no longer than a few months. Data for a project are derived from corporate databases, claims data, or external sources.

We have a large number of disk drives (20 writeable disks at the moment comprising 236 Gigabytes of space, with another 60 read-only disks on our system and at least a hundred disks on other systems). As disks have been added, it has become more and more difficult for our users to keep track of where their data and programs are stored. The problem is increased by the occasional need to move a project's files from one disk to another for operational reasons. In the past, most users had hardcoded file locations in their programs; when a project was moved, every program referring to its programs or data would have to be located and changed. For projects whose results were used by other projects, this could be quite a task.

Our solution to this problem is project logicals. Each project is assigned a unique identifier (4-8 characters, following SAS library naming conventions).

The StdOCN project (for standard ad hoc reporting requests on Outpatient Care Network providers) has the following logicals defined in mt$projdefs:stdocn.com
:

```
$ ! Provider counts
$ define  stdocn-pgms  -
                asreshome:[projcode.stdocn]
$ define  stdocn-data  -
                asqc:[projdata.stdocn]
$ define  stdocn-init  -
                stdocn-pgms:stdocn.sasinit
```

A program (or person) that wants to refer to this project's programs would use STDOCN-PGMS in place of a drive and directory; STDOCN-DATA would refer to the project's data.

The STDOCN-INIT logical points to a file containing SAS startup code for the project. This code typically contains a LIBNAME statement to define the main data directory for the project, but can contain any other SAS code needed by the project; it might include other initialization files, define macro variables or libraries, or set SAS system options.

PRODUCTION LOGICALS

We also have "production" data which are derived from corporate INGRES databases. We have set up a three-tier system of logicals for references to these data. The top level, PRD, is the level which most programs will use. The second tier, STG, is used during our updates of the data. STG data may not be complete until all processing has been completed, and so are not suitable for normal use. The third tier, DEV, is used to store data sets under development (a new table being added, for example).

In the initial discussion of OpenVMS logicals, I mentioned that a logical can refer to one or more than one file system object. This capability is exploited in the logicals for production data. Look at the logicals defined above:

```
$ define  dev-core-xtrn-data   -
                    dev-core-xtrn-data-w,
                    stg-core-xtrn-data
$ define  stg-core-xtrn-data   -
                    stg-core-xtrn-data-w,
                    prd-core-xtrn-data
$ define  prd-core-xtrn-data
                    prd-core-xtrn-data-w
```

The SAS initialization file executed by all users contains the line

```
%include
  "%SYSGET(my_datalevel)-mtbazaar-libnames:";
```

By default, the value of my_datalevel is PRD, so the line above would resolve to

```
%include
  "%SYSGET(my_datalevel)-mtbazaar-libnames:";
```

which would (after a few more redirections) point to `prd-core-xtrn-data-w`, a single directory containing PRD-level data sets.

If the user had set my_datalevel to be DEV, the ultimate resolution would be to three directories, which would be searched in sequence. If a data set had been created in the DEV directory, it would be found first. If it had not been created in DEV, the STG directory would be searched. If it were not there either, the PRD would finally be searched.

This system of multiple logicals allows us to have the equivalent of separate production, staging, and test systems, without having to create and maintain three separate copies of the data. It has turned out to be very convenient for program development.

SAS SETUP

One line of the common login is a call to a SAS setup procedure. It's too long to show in its entirety, but here are a few lines:

```
$ define sas$user  -
                my$root:[sasuser]
$ define sas$init  -
                mt$comsas:autoexec.sas
$ define sas$news  -
                mt$comsas:sasnews.txt
$ define my$sasconfig  -
                sas$user:sasconfig.cfg
$ define my$sasinit  -
                sas$user:autoexec.sas
$ define my$sassession  -
                my$root:[sasuser.session]
$ define sas$config  -
    my$sassession:pid'f$getjpi("", "PID").cfg
$ if f$mode() .eqs. "INTERACTIVE"
$ then
$    copy /nolog   mt$comsas:sasconfig.all, -
          mt$comsas:sasconfig.interactive, -
          my$sasconfig sas$config
$ else
$    copy /nolog mt$comsas:sasconfig.all, -
          mt$comsas:sasconfig.batch, -
          my$sasconfig sas$config
$ endif
$ !
```

Some things to note here: the SAS initialization file, configuration file, and news file are all diverted from their standard locations to department or user locations; interactive jobs don't use the same configuration file as batch jobs; and a configuration file is created for each session in order to provide a way for each to specify their own set of configuration options if they so chose (user exists).

SAS$USER

The SAS$USER logical points to the SASUSER directory. We set it up under the users home directory.

SAS$INIT

The SAS$USER logical points to the SAS initialization file in a common directory. Here's a simplified version of what that looks like:

```
options sasautos=
  ("%SYSGET(my_datalevel)-base-baza-macs:"
    %SYSFUNC(getoption(sasautos)));
%include
  "%SYSGET(my_datalevel)-mtbazaar-libnames:";
libname default '[]'
  cachesiz=65024 alq=17 deq=170;
%put INFO: Libname MTBAZAAR refers to the
%SYSGET(my_datalevel) Metrics Data Model.;
%include 'my$sasinit:';
```

SASAUTOS sets up the autocall macro libraries. Note the use of %SYSGET(my_data_level) to bring in the data tier that the user wants (PRD, STG, or DEV). The second line defines the data libraries). The third line sets up a SAS libname for the current directory (the one in which the SAS session was started). The next to last line puts out a message reminding the user what their data level is, and the last line provides a user exit – it includes the autoexec.sas file created by the user in their SAS$USER directory. This will ordinarily be empty.

SAS$CONFIG

The SAS configuration file, which set system invocation options, was the trickiest to set up. Unlike SASINIT files, the configuration file can't do a %include. The only way to have both a standard department CONFIG and a user CONFIG was to create a new configuration file for each login session; it's just a concatenation of the two files. There was a similar problem with batch versus interaction configurations.

This is what the interactive configuration file might end up looking like:

```
   /bufno=2             /catcache=2
   /cbfn=2              /cc=fortran
   /compress=yes        /errcheck=strict
   /errors=5            /fsdevice=dectermc
   /fullstimer          /linesize=132
   /logmultread         /msglevel=i
   /noovp               /nosymbolgen
   /pagesize=55         /rsasuser
   /ssize=819200        /stack=80000
   /work=sys$scratch    /wrkcache=65024
   /yearcutoff=1920     /fsdevice=dectermc
   /altlog=my$sasaltlog
   /frms=land           /noerrabend
```

Some entries to note: `/ALTLOG=MT$SASALTLOG` sends a copy of the SAS log to the specified file. This provides a backup when you accidentally type ENDSAS without saving your work. `/RSASUSER` forces the SASUSER file to be read-only, allowing multiple sessions to run at the same time. `/WRKCACHE=65024` causes the WORK directory to be cached with the largest possible cachesize. `/FSDEVICE=dectermc` sets a default terminal type.

TERMINAL EMULATION

Making our terminal emulator program (KEATerm) work correctly, consistently, and in an obvious manner with both SAS and the OpenVMS system editor was easily the most difficult part of creating a common user environment.

Under version 6, SAS supports three interfaces: line mode, in which you enter one line at a time; full screen mode, which uses addressable ASCII terminals such at the DEC VT-100 or its emulators; and X Windows, which requires a X terminal or emulation package.

The first of these methods, line mode, is not really practical to use. There's no program editor, and full screen procedures such as FSEDIT and FSVIEW can't be used.

The third method, X Windows emulation, also isn't practical. Besides the problem of emulator programs being expensive, running the SAS Display Manager is just too slow to be practical. This is partly because our OpenVMS machine is remote, not local, and partly because SAS Institute chose to use the Motif interface, which is, in the words of the X Speedup FAQ, "an absolute pig".

That leaves the second method, full screen mode. In version 6.12, SAS supports a number of terminal and terminal emulators, mostly variations on the old DEC VT-100 terminal. We tell SAS that we are using a DECTERMC terminal (Color DECTerm emulator), and we tell our terminal emulator program to behave like a DEC VT-420 terminal. This gives us color support in interactive SAS and in those OpenVMS apps that support it. We had to create a custom keyboard mapping in KEATerm to create the mapping we wanted (basically, to emulate standard Windows key functions rather than DEC key functions, since no one here was even seen a real DEC terminal in years). That meant we also had to create a custom key mapping for TPU, the DEC system editor.

In a previous version of the common login, we used a custom full screen device created with PROC FSDEVICE. This made the keyboard mappings much easier. Unfortunately, when we wanted to make some additional changes to the custom device, we discovered that PROC FSDEVICE is no longer supported by SAS Institute, so we decided to stop using it.

EFFICIENCY CONSIDERATIONS

SAS 6.12 has several system-specific features to increase the performance of SAS programs.

CACHING

Caching allows multiple pages in a SAS dataset to be written or read at once, reducing I/O. Cachesize can be specified at the data set level:

```
   data cache.big (cachesiz=65024);
```

or at the library level:

```
   library stdocn 'stdocn-data:' cachesiz=65024;
```

It can also be set for the WORK library using the WRKCACHE option at system invocation or in the SAS configuration file:

```
   /wrkcache=65024
```

Use of a cache does not typically reduce execution time by much, but does result in reduced I/O. Running this code:

```
   data cache.test1;
      length a b c $100;
      retain a b c ' ';
      do i = 1 to 100000;
         output;
      end;
   run;
```

had these results when CACHE referred to a cached library:

```
   Buffered IO:        175
   Direct IO:          664
```

and these results with an uncached library:

```
   Buffered IO:        181
   Direct IO:         1132
```

CPU and elapsed times were approximately the same, but I/O was almost halved.

Note that the name of the system option drops the final *e* in *cachesize*.

READ-AHEAD AND WRITE-BEHIND

After you have set your library to use caching, you can specify read-ahead and write-behind for individual datasets. Read-ahead tells SAS to read more of the data set than it has immediate need for. The result is decreased execution time, since some of the data will already be in memory when SAS calls for it. Read-ahead is specified with the RAH=YES data set option, and write-behind is specified with the WBH=YES data set option.

It is almost always appropriate to specify WBH=YES when writing a sequential SAS data set (I have not experimented with in-place updates). RAH=YES is appropriate when you are reading a data set sequentially. It might not be appropriate for reading a data set using an index, as some of the read-ahead I/O will be wasted when the read-ahead data aren't used. For jobs which are run often, it would be a good idea to run some test jobs with and without read-ahead turned on.

Use of RAH= and WBH= can result in amazing reductions in elapsed time. Another department saw the elapsed time for one job go from 8 hours to about 40 minutes. My department has not seen a reduction of that magnitude, but a halving of elapsed time is not uncommon.

RAH and WBH cannot be specified on LIBNAME statements; they must be specified on each data set.

FILE ALLOCATION

OpenVMS allocates files in blocks of 512 bytes. By default, it allocates 17 blocks at a time (this value may be different on your system). You specify a larger number using the ALQ= option for the initial allocation and the DEQ= option for secondary allocations. Specifying ALQ= and DEQ= appropriately can result in significant reductions in I/O and elapsed time.

MULTIBLOCK COUNT

The options above apply to SAS datasets. Another option, MBC=, specifies the number of buffers used when writing external files. You can specify MBC= in a FILE or FILENAME statement. It can also be specified in DCL using the command

```
   set rms /block=32
```

(or whatever value is appropriate for your system).

INSTALLED IMAGES

Another OpenVMS feature which has the potential to decrease execution time is the use of installed images; which reduces startup time by preloading commonly used executables (roughly equivalent to the use of the LPA in MVS). SAS Institute recommends the use of installed images when when two or more users are running SAS simultaneously. Although we meet this criterion (there are currently 9 concurrent executions; sometimes there are as many as 40), we have chosen not to implement installed images because of the difficulties they cause with maintenance. We plan to rethink this decision later in the year.

THE FUTURE

What's the future of OpenVMS, and of SAS under OpenVMS?

It's hard to say. There are constant rumors that Compaq plans to drop support for Alpha chips, or for OpenVMS. Compaq always denies these rumors. OpenVMS on Alpha machines is stable and fast. It can handle large amounts of memory and disk space easily. DCL is a bit of a dog compared to Perl, REXX, and other modern scripting languages, but it does get the job done.

Whether OpenVMS will continue to be a viable platform for SAS Software is a different matter. It appears that SAS Institute has decided to relegate OpenVMS to a second-tier OS. The removal of VTxxx terminal support in Version 8 is a severe problem for us. Enterprise Guide would be a good alternative, but the server does not run under OpenVMS. Using X Windows emulators would be expensive, and worse, very slow. But we don't want to run everything in batch mode. At this writing, I don't know what we're going to do.

REFERENCES

Compaq Computer Corporation, *OpenVMS documentation website*, http://www.openvms.digital.com:8000/index.html

Mulder, Art (editor), *comp.windows.x: Getting more performance out of X. FAQ*, http://www.faqs.org/faqs/x-faq/speedups/

SAS Institute Inc., *SAS Companion for the OpenVMS Environment, Version 6, Second Edition*, Cary, NC: SAS Institute Inc., 1996. 527 pp.

TRADEMARK CITATION

CONTACT INFORMATION

Jack Hamilton
First Health
750 Riverpoint Drive
West Sacramento, California 95605
JackHamilton@FirstHealth.com
http://www.qsl.net/kd6ttl/sas/sas.htm

Selected papers are also available at:

http://www.sashelp.com/Articles/ViewPapers.asp