Welcome to the Advanced System Manager's Console. (Advmgr)

****
TABLE OF CONTENTS
****

****

1. Introduction

****

I have been an advocate of the VMS (later OpenVMS) operating system since I was first introduced to it back in the early eighties. My computer experience goes back to the early seventies while with the Scientific Research Staff of Ford Motor Company. VMS was, and many of we old timers think still is, the best general purpose time sharing operating system ever written. I shall not sing its praises here because you already know this. What I have been doing for the last few decades is developing software to assist a VMS system manager in doing just that. I was the system manager for a software company a long time ago and found it challenging because I did not know enough at the time to develop the kind of software that I and other system programmers have developed over the years. I wish I had then what I have now and would like to pass it on to you.

****

2.   Acknowledgments

****

This package contains software I've put together over many years to assist in managing VMS and in developing so called "System Programs". Some is my own and much is based on public domain software written by Brian Schenkenberger of Tmeis and Hunter Goatly formerly of the University of Kentucky. I owe a debt of gratitude to them, Brian who has helped me numerous times to understand what the software was doing and getting it running, and Hunter who has not only inspired me with his writing in the field but has done much to promote VMS.

I'm also indebted to Ralf van Diesen of Advanced Virtualization Technologies in the Netherlands. He has most generously given me his software and helped me set it up so that I could run a two node AXP cluster and write these programs to work on it. He has been tireless in zoom sessions despite the daunting time difference and in responding to a deluge of emails. He has become one of my best friends. His software is the best for running AXP VMS.

Much of this software belonging to the above mentioned people has been improved slightly by making it a bit easier to use. For example, many programs require the PID of a process to work. I've modified things so that the terminal name will also suffice. Much of the "internal workings" of these programs have common sections and these have been moved to a shared image (MARLIB), saving memory and disk space while improving reliability. I have also re-written most of the programs using MACROS that make MACRO32 and MACRO64 fully structured languages.

****

3. Legal

****

This software is provided "AS IS" and is supplied for informational purpose

Note!
Most of this software runs in a highly privileged mode and assumes you are running
with all privileges. It runs at high IPL and some loads code into the operating
system. If you run any of this software and something evil happens, you have been
warned.

****
4. Advmgr Menu operations descriptions
****

If you are using the Advanced Management Console, you need to know a couple of
things about how I've written stuff.

First off, I try to be minimally intrusive on your computer system. I do not
copy stuff to your system disk. Installed shared/privileged images are installed
from where you place the ADVMGR directory tree (hopfully not the system disk).
The same for execulet images loaded into the operating system at boot time. The
system manager must reference ADVMGR:[COM]ADV_SETUP.COM in
SYS$MANAGER:SYSTARTUP_VMS.COM and follow a few other requirements such as:

Many of my scripts and my MENU program call other scripts and/or images in the
ADVMGR directories. They do so using logicals which are defined in LOGIN.DAT
using the installed program, LOGIN.EXE called from the LOGON.COM file. In
addition, I use an unusual way to define default directories in the user
authorization file.

The default device/directory in the UAF should be based on a concealed logical
defined at system boot in the system table,
define/sys/exec/translation=concealed, which points to the physical device and
root directory on the system for each user. An example should suffice. On my
system, ADVMGR is defined as $2$DKA100:[ADVMGR.} which is defined in
ADV_SETUP.COM referenced in the system startup script. In the UAF record the
default device and directory are ADVMGR:[000000]. This should be done for any
user whose programs need to be run by other users on the system. I do this for
developers, myself and ADVMGR.

This has two uses. First, it makes changing the physical disks much easier. Just
change the concealed logical. Secondly, it allows the users of my software,
including LOGIN.EXE, to do things without modifying code/scripts. Defining your
device and root directory as a concealed logical makes this information
discoverable and can be used by LOGIN.EXE.

However, this is not mandatory. Other users can make use of LOGIN.EXE and their
own LOGIN.DAT files while using the normal way of defining default device and
directory for the UAF. LOGIN.EXE recognizes this and creates a concealed
logical, Z$Z$, in the JOB table for use in further definitions in LOGIN.DAT.

Furthermore, a user can override the internal definition of Z$Z$ by defining it in LOGIN.DAT as the very first definition in the file under the search logical definition section. You could do this for new developers by defining it as your concealed system logical so they would have access to your material to start. Overrides in LOGIN.DAT can be added as needed by the new user.

In referencing my material, I use logicals that point to actual directories and follow the naming convention of using the extension of the files contained therein followed by the letter "D". To use my material, the minimum logical definitions are: EXED, COMD, DATAD and TEMPD. These are defined for you if you are using the Advanced Managers Console and logged in as ADVMGR. The program, EXED:LOGIN.EXE, uses definitions found in SYS$LOGIN:LOGIN.DAT. You probably will also want to define HELPD and MLBD plus any that you think will be helpful to your operation. As an example, assuming you are logged in as ADVMGR, the logical MARD is defined in LOGIN.DAT as though you put the command "$ DEFINE/JOB MARD ADVMGR:[MAR]". The actual specification for this directory on my system (not using concealed logicals) is: DKA100:[ADVMGR.MAR]. The LOGIN.EXE program will take a full file spec passed as an argument, if you want to experiment with another file of definitions.

LOGIN.EXE defines a list of things specified in SYS$LOGIN:LOGIN.DAT:

Logicals in the JOB table
Search Logicals in the JOB table
Logicals in the USER table
Symbols that do many thing for you
Color symbols needed to colorize our operations
Key Definitions
Default Protection
Process name (optional and a bad idea if you use more than one login)

If you are running MENU on a cluster, you will notice that some menu entries are followed by a red asterisk. These can be run on other nodes from the one you are logged into. Some of these have violet descriptions rather than green indicating they are cluster wide by their nature. Those entries which are in violet do not use SYSMAN. Green entries followed by an asterisk, do. If the asterisk is gold in color, it means that the output from that procedure can be large and will be written to a file defined by the cluster table logical ADVOUTPUT. This is discussed in detail below.

Regardless of whether you are on a cluster, an entry who's description is white indicates that it must be run at the same level (not as a sub-process) as MENU and thus will terminate MENU when selected. The keypad "0" key will restart it.

The data file that drives MENU is constructed as described in the file, DATAD:ADV.DAT. Here you can learn all the little details of how the programs/scripts are displayed and executed.

In order to enhance readability and to be "cool", we have colorized much output from our programs/scripts. In addition, due to the nature of the output from certain operations and the fact that you may be running the operation on a large system and/or multi node cluster, output in some cases can be staggering. If you are logged into a VMS system over a browser (as you might be using AVT's AXP emulation), color does not work and scrolling is a guess. We suggest you use a terminal emulator (we recommend PUTTY which is free) and come in over telnet. PUTTY and other terminal emulators are capable of buffering a lot of output and this can be used to advantage.

When the output exceeds the number of lines on your screen, see if your buffer is large enough to scroll back to examine it from the beginning. If not, you might increase the buffer capacity of the terminal emulator. Too large a buffer takes up resources on your terminal emulator host system and will, at some point, make scrolling an unpleasant task.

In this case, we suggest diverting output to a file. This is done for certain programs by your having first defined a logical, ADVOUTPUT, into the cluster table (which you have even if you are not running a cluster) as a full file specification. e.g.

```
$ DEFINE/CLUSTER/EXEC ADVOUTPUT ADVMGR:[TEMP]ADVOUTPUT.TXT      ! To create the
logical
$ DEASSIGN/CLUSTER/EXEC ADVOUTPUT                               ! To remove it
```

Do this with care as a bad file specification causes all hell to break lose. If you use concealed logicals in the specification (as above), be sure they are defined the same in every node’s system table.

When ADVOUTPUT is defined, the image/script uses this to write the output to the file and not the screen. Programs using this are written to eliminate the colorization which would make the file very difficult to read. In order for this operation to work on a cluster, assuming you select multiple nodes on which to run the process, the output file you have defined is opened in append mode. It is your responsibility to delete/rename this file at appropriate times in order to eliminate the “old” information.

In addition, the output file defined in ADVOUTPUT is opened on a node determined by SYSMAN. If I am running on my NODE1 and define the output file using the logical TEMPD:, it appears on NODE2. If I define it using R$:[temp[, it is written on node1. You will have to check around to find out where it is.

To make life a little easier, we have a menu entry that defines/undefines ADVOUTPUT in such a way that it is wriiten on the node on which the program(s) are to be run. In addition, GOLD-KP0 will "$ TYPE/PAGE" the output file and BLUE-KP0 will delete it. How you view the file may easily be changed in LOGIN.DAT. GOLD-COMMA on the ten key pad will deassign ADVOUTPUT.

Note!

The MENU program is set up to issue a CTRL-D to the running TSR program (see below) in order to display a calendar. We feel this is a nice touch and makes the menu display a bit more useful. It will do this once a minute so the display is current + or – one minute. The TSR is started from ADVMGR:LOGON.COM.

Many procedures ask for argument(s). Though brief, the request is designed to be descriptive. In most cases, only one argument is needed. In those instances where more than one is required, there is a code. (2sp) means two arguments are needed and they should be separated by a space. (2,) means they should be comma separated. For requests for pid or terminal name, get this information from the display of "Show Processes". Pid is hexadecimal and terminal name should leave out the ':'.

Below are descriptions of menu operations in the ADVMGR Menu. Please note that the selection 2 - 4 are new menus in themselves with many choices including a discussion. Read the discussion first as it not only describes how things work, they point you to literature you need to read to understand what you are doing.

#---
"Show Processes" - This selection runs a program that produces output somewhat similar to "SHOW SYSTEM" but is more brief and includes the terminal name for those processes that have one connected. It will request an argument of any

letter (rather than just return) to limit output to those processes that have terminals connected only.

#---
"All Security" - One of the prime responsibilities of a systems manager is security. Please read the security manual. There are eleven categories of objects that allow security related settings. This script will be of considerable assistance in assigning permissions, creating identifiers, granting identifiers and assigning them to objects to control access to them in complex ways.

To modify a user, we have written a program to assist and make it easier. It presents a screen a bit like that of AUTHORIZE showing a user. The labels are numbered like a data entry screen so you can select and modify them. It is a bit involved with multiple screens and different ways of entering data to fit the many different formats of the fields in the UAF record. Please read the details at the end of this program in addendum A.

You will run the AUTHORIZE utility through this script as well as the ACL Editor. (I can never remember the required default directory to run AUTHORIZE). In addition, you may set/remove the current privileges for any process on the system, including your own, regardless of the authorized privileges assigned to that user. This assignment will last for the duration of the process. You have control of the process priority through this menu and can encrypt/decrypt any file.

Additional information and suggestions are available from that menu.

#---
“Queue Management” – Queue management is complex and involves multiple DCL verbs and many switches for each command. We have written a script that will assist the manager in handling normal queue operations. Operations are grouped (except for queue manager operations) by DCL verb. Switches are displayed and numbered and an option exists to display DCL HELP for each of them.

Based on your input, a command is formed and you are given the option to actually execute it.

#---
Opcom - All VMS installations save for the very smallest can benefit from having an operator in the computer room who can mount tapes and disks and handle the printers. To facilitate communication with said operator(s) OPCOM.EXE was written by DEC to allow normal users to communicate requests to the operator(s) and receive replies.

Normally the consol terminal named OPA0: is enabled as a receiver of OPCOM messages. Other terminals may be enabled to do the same. Functionality may be destributed among OPCOM terminals. For example, one operator terminal bay receive and handle requests related to printing, one for mounting/dismounting tapes/drives and a third for everything else. There are a variety of classes for which an operator terminal may be enabled and to which a normal user may designate a request.

As in the Queue management script above, help for each switch (REPLY command only) is available.

Based on your input, a command is formed and you are given the option to actually execute it.

#--
“TCP/IP Operations – This handles TCP/IP configuration, stop/start, Time Synchronous operations and a few other things in addition to running TCPIP.

#--
"Roll Your Own". Trial and error. Uses SYSMAN. Good place to add your own operations for ADVMGR. Test them this way and add them to the menu later.

#--
"Suspend/Resume a Process" - Suspend/Resume will either suspend a process or resume it, depending on its present state.

#--
"Kill an RWASTed Process" - Special Process Delete is more than the DCL "STOP PROCESS_NAME" or "STOP/ID=XXXXX". STOP will not work if the chosen process is in resource wait state. However, this method sneaks into the process' PCB table and resets the resource wait bits prior to deleting the process.

#--
“Issue Remote Commands” – This operation will allow you to issue a command(s) to any other terminal on the system. You may do this to a single terminal, a list of terminals or all the terminals on a node. Commands may be a single one line command or a DCL script file. A command beginning with a tilde, “~”, will cause a CTRL-Y to be issued before the command.

#--
"Show Other's Default Directory" - Other's Default Directory will use an executive image to determine the process' default device and directory.

#--
"Show Others Commands" will print out the full command recall buffer of the other process. It is useful for helping others.

#--
"Show Disk Free Space" - DEC/COMPAQ/HP never thought that a knowledge of the percentage of a disk that was available (free) was of any interest to a system manager. We do. This script lists all mounted disks on the system and shows the percent free space on each one.

#--
"Show Open Files On a Device" - This will show you who has what file(s) open on a drive. If it’s the system drive, you will get a bunch of output.

#--
“Show Who Has a File Open” was written by Edward A. Heinrich some time ago. He did an excellent job but had to limit the number of processes to display. Not a problem for most installations but might be for large operations where lots of folks might have a file open at the same time. Edward also included more information than I felt was appropriate. I have a love/hate relationship with RMS so the initial work on the file specification is done in DCL which provides a valid device and file ID to my MACRO32 program that searches through a bunch of control blocks to get the info.

I determine the total number of process slots that are generated into the system and allocate P2 space to store information. The number of process slots is guaranteed to exceed the number of processes on a given node that can have a file open and P2 space is damned near infinite.

#--

"Show the Lock Holder" - Get lock holder takes the name of an IASM file, the specific key for that file's record which is locked, and determines the name of the process which holds the lock. This is useful for finding out who got the lock and went fishing. You can free the record by deleting the holder's process.

#---
"Clear a Device Error Count" - Does just that plus memory and cpu error count. Especially useful for device driver development.

#---
“Force a Dismount” - Sometimes users allocate/mount a device for their personal use and then forget about it. I know from years of being a system manager that they don't even log out at the end of the day. This program will take a full device name ($2$DKA101) as input and force the dismount and/or deallocation of the device. It will do it cluster wide so you will likely get some error messages about not being found on a node.

#---
"Monitor" - Monitor is often run interactively. It displays a bunch of information in an easily read graphical or list display in real time. In many cases, you need to monitor for information over a long period of time. The script produces a sub-process (/NOWAIT) that stops when you tell it to stop, creates a file called ADVMGR:[TEMP]MONITOR.SUM that you can look at and notifies you when it's complete. Don't log off when it's running or you'll stop it. You also have the option of running the procedure as a detached process. No notification but you can log out.

#---
"Image Accounting" - Accounting is enabled by default. However image accounting is not. This is because it places a very heavy load on the operating system. Do not run this procedure for long periods unless you're willing to field the complaints.

This procedure creates a sub-process which will enable image accounting for the delta time you specify and then produce a report of the results in ADVMGR:[TEMP]ACT.LOG. It takes two arguments, the delta hours and the delta minutes.

#---
Run AUTOGEN - AUTOGEN is the DCL script where permanent changes to system parameters are made. Use it because it checks your proposed changes against system requirements and prevents you from generating an unusable system. NEVER run the AUTOGEN script with REBOOT as the final parameter in the command line. This is especially true in a cluster environment as changes could be incompatible with nodes not modified.

Instead, read the report AUTOGEN creates. It is full of information including errors that can be corrected prior to the commitment of re-booting. Our script does this.

#---
"Show Working Sets" - It is often desirable to know about each process' use of working set pages. This selection will type out a list of all processes and information on their working set use and parameters.

#---
"Login as Other" - Login as another user does not require that user's password, just the user name. It can be used to test a program in another environment.

#--
"Display Instructions" - Displays a brief note referring to this file.

#--
"Date and Time" - This will give you the date and time plus a bunch of other information and in living color. I run it at login.

#--
“lock Screen” – You may wish to step away from your terminal without having to bother to log out then back in and restart a complex editing session. Select this option and enter your password when you return. Anyone can use it.

#--
"Spy" - Monitor another terminal on this node. Discussed in detail below.

#--
“Log Off” - LOGOUT

****
This is the end of the menu description.

****
5. Advmgr additional programs
****

We have included a few other goodies that are not run from the menu.

#____
“Shut” This is an improved method of shutting the node/cluster down. It is a DCL script, TSHUT.COM which is invoked with the command “shut”, defined for you in LOGIN.DAT. It gives you lots of options, including the ability to change the EXPECTED_VOTES in a cluster, if needed. It makes sure “REBOOT_CHECK” is always part of the operation and works in both a cluster and single node environment.

If invoked from an administrator logged in over telnet (any way that involves TCP/IP), the shutdown proceeds to the point that the system ceases to function but stops the process of shutdown before it is complete. Forceful stopping of each node will be required.

On the other hand, invoked from the operators console, the shutdown is clean and complete.

#____
“TSR” TSR.EXE is Paul Klisner and my "Terminate and Stay Resident" program that sits up in P1 space where it stays as long as the process remains. It captures control keys (out of band keys) and executes code asynchronously to other operations in the same process. Thus (in its present form) a CNTRL-X will clear the screen, a CNTRL-D will write a nice looking calendar in the upper right corner of the screen with today's date highlighted and a CNTRL-V will toggle the DCL verify state even in the middle of executing a script. Can be very useful when debugging DCL scripts.

CNTRL-k will change your privileges, toggling between the default privileges in the user User Authorization Record and the Authorized privileges. This can be useful if a user is set up with "normal" default privileges which will keep him from accidentally damaging the system, but will allow a trusted user to gain his fully authorized privileges with a single keystroke and then set things back to normal when he is done. For this to work, the user must have the default privilege of SETPRV.

Note: The calendar display now includes a toggle status display on the bottom left. A V for verify on, a P for authorized privileges in effect and an A for ADVOUTPUT defined.

A new addition is the ability to capture the file id'S of all the files a process has open at any given instant. It does this with the CNTRL-F key. Rather than pour file ID's all over your screen in the middle of a process, the CTRL-F key appears to do nothing. However, it actually creates a logical (RICK_FID) that contains all the file ID's. When you are back to a command prompt, FID_TO_NAME.COM will translate all the file ID's to full file specifications and print them on your screen, or put them in a file (TEMPD:OPF.TXT) if you pass the script an argument.

Please note that this operation gives erroneous results when you use it from a sub-process. It actually gives the results for the parent.

CNTRL-G will switch the undocumented and unsupported DCL command "SET WATCH FILE" which documents all the file operations happening in your process. This is a toggle.

Finally, a CNTRL-J will print out the startup greeting message (instructions) on your terminal.

We have also made things more flexible by allowing changes in what control keys are associated with what operations and even what operations are enabled. You control this by defining a symbol to run TSR.EXE and passing it an argument. The argument consists of a string of characters without spaces. It consists of variables separated by commas and must be in pairs. The first item in the pair is a number representing the operation as displayed when TSR is run or when you hit CTRL-J (^J). The second item in the pair is either a letter representing the control character which will be trapped and will activate the function described or a null (two commas in a row without a space) which will disable the function and not trap the control character associated with it by default. ^J will show the results. An "^" followed by a space indicated that the function is disabled. If you change ^D for the calendar, the calendar will no longer appear in the MENU display unless you change the command line given to the MENU program.

#____
"Spy" SPY.EXE Run it from the menu or

```
$ SPY :== $EXED:SPY.EXE
```

At the DCL command line, type:

$ SPY YYxn[,letter,letter] where YYxn is the name of the terminal to be monitored. Optionally you may specify single letter arguments for the two control characters used; the first for disconnect and the second for command entry. For example:

```
$ SPY TNA4,f,b
```

Brief instructions will be printed and from that point on, whatever appears on the screen of the monitored terminal will appear on your terminal as well (with exceptions noted below).

By default, CTRL-\ will decouple you from the other screen and exit the program, returning S0 space to the OS.

CTRL-B will prompt you to enter one DCL command for the other terminal (process) to execute.

TERMINAL EMULATOR SETTINGS:

We use Putty. Spy works best with the following settings that seem to be different from the default:

```
Settings: IP and Port as appropriate. Con type: other - telnet
Keyboard: (127), ESC[n~
Window: 132X40, change size of font, the rest as you like.
Translation: ISO-8859-1:1998 (Latin-1, West Europe)
Connection: Disable Nagle's algorithm, IPv4
Telnet: BSD, Active
```

BACKGROUND:

SPY is a labor of love. We did it over a period of more than six years and without the benefit of driver listings. Thus it was done by means of deductive reasoning, "poke and hope" and a terrifying number of system crashes.

Along the way we detoured down a variety of dead ends and made some wrong turns, but, in the end, we have something that we feel is useful, but not perfect. It seems to handle any terminal driver based on the TT class terminal driver and can be run by a person logged in over DECnet. It handles large amounts of output without dropping characters and seems to handle escape sequences generated programmatically very well. Command line editing works as advertised but does not always display correctly which can make it a bit frustrating, especially to first time users.

Other versions (commercial) may have overcome problems we are still working on. We have no experience with them.

KNOWN DEFICIENCIES:

One of the problems we have encountered right from the beginning is the propensity of the Alpha class driver to omit a leading character from the output stream placed there at the beginning of a $QIO. It took some time, but patterns started to emerge and we have been able to compensate for this.

So far We've tested: OP, TT, VT, TN, FT, RT (as your terminal) and we suspect it will work on TX. There is one known exception; while you may monitor someone by doing a SET HOST to their node, you cannot monitor a person who has logged in over DECNET. Much to our surprise, doing so did not crash the system, it just didn't work.

Those programs which utilize the SMG$ Screen Management Facilities, including DECTPU, will not display their buffers on the monitoring screen.

If you come up with a situation where you have a terminal device name that uses the class TT driver and SPY tells you it's a bad choice, call us. We'll make arrangements.

Command line editing can be mildly frustrating. You will notice that things sometimes echo strangely on both terminals. This is most notable deleting characters with the backspace key. It echos the state of the command line on a new line and, on the terminal being monitored, will not overwrite (delete) the first character entered. Actually it does and full line editing (arrow key movement, CNTRL-R and toggle over-strike) all work just fine. If it doesn't echo exactly right, CNTRL-R will re display the actual command line.

We're working on it. You may discover other problems. It's complicated.

#____
Command line prompt

Many operating systems employ part or all of the default location where the user is working as the command prompt. VMS doesn't. It should.

You have a program from Brian Schenkenberger that will intercept calls to SYS$SETDDIR and modify the prompt to be the new directory structure limited by the size of the prompt area in the OS. It uses a modified execulet that captures the calls to the system service, allows a user written segment to execute before continuing on to execute normally. It is placed in SYS$LOADABLE_IMAGES by KITINSTAL.COM when you installed it and loaded with a call to EXED:SSDDLOADER during ADVMGR setup at boot.

#____
EVE Editor as a sub-process. You can start the EVE editor as a sub-process that will allow you to switch between editing a file and the DCL parent process with a single keystroke. Start the editor with:

```
@comd:sedit.com.
```

Edit a file with:

```
$ e filename.ext
```

The routines will edit it straight away, if it’s in your default directory, search for it in many places and ask if you want to edit what it found and allow you to create and edit a new file in the default directory. Very convenient. The way I have things set up, you can switch back and forth between editing and the DCL parent process with the keypad “Enter” key.

Below are my key definitions. Operations following the "/" are the GOLD operations. It should serve you as a starting point.

```
            EVE Editor
(ten key pad)
Find/Find Wild         Find Selected     Find Next   Do
Select                 Del L/Rest L      Cut         Replace
Start of Line/Open Line Del Char    Copy
End of Line/Mark  Write Buf/Go To   Paste
Next Buffer                  Gold        Attach Parent/Sh Key

(six key pad)
(insert)               Top(home)   Page UP
Delete Selected/Restore(delete)    Bottom(end) Page Down

F1 = Help
F2 = Set Right Margine 80/132
F3 = Uppercase Word/Lowercase Word
F4 = Quit
F5 = Fill Paragraph/Capitalize Word
F6 = (Leave this alone and don't use it)
F7 = Show Buffers/Buffer Messages
F8 =
F9 = Save Attributes
```

F10 = One Window
F11 = Next Window
F12 = Two Windows

You can redefine keys to your own purpose by editing EVE_KEYS.EVE in the .EVE] sub-directory. Then Start EVE by typing 'EDIT' at the DCL command line. Hit the DO key, PF4, and at the editor command line type: @ADVMGR:[EVE]EVE_KEYS.EVE. This will load and execute the EVE commands in that file. This file defines your keys. Next, issue the command: SAVE ATTRIBUTES, (F9, using my definitions). This will save all the changes into the binary section file which is loaded at startup.

#____

Compile/Link. I have included a script, COMD:BUILD.COM, which can build most anything written in any language supported on your system. The script can easily be modified to include additional languages. It is very easy to use but requires the above mentioned logicals. Type “build” at the command line to see instructions.

Type “BUILD M DISMOUNT” to see it in action. It will compile and link MARD:DISMOUNT.MAR and put the executable in EXED:.

#____

Login hook. Most installations connect to the internet, even though this is a very dangerous thing to do and should be avoided. At a minimum, secure your system from remote login by using the LOGINOUT hook mechanism. If you have a “C” compiler, you can modify ADVMGR:[c]LIOH.C to suit your needs. As presented, the hook will prevent any login from TCP/IP (Telnet, SSH) that does not begin with “USER”. On my system all accounts beginning with “USER” are captive.

#____

SHOW_CLUSTER. The ten key pad key, comma (or in some cases plus) will invoke the command SHOW_CLUSTER/CONTINUOUS. Its initialization file is DATAD:SHOW_CLUSTER.INI which you can modify to suit your own purposes. Panning is enabled. Read the details in System Manager Utilities 2.

#____

SEARCH - The DCL command verb SEARCH is an often used facility that most everyone finds helpful. Like most other DCL verbs, it has many switches. We concidered treating it like other verbs where there is a menu selection and all the switches are displayed and you can get help.... That is not how most programmers use it.

What I usually want is a quick reminder of what programs might have a specific system service call to remind me of the named parameters so I can use the same thing in the program I'm writing now. Under the premis of KISS, I have writted a script that simplifies using the search verb without eliminating all of the most useful switches.

SEA is defined in LOGIN.DAT to execute RSEARCH.COM. This script takes five parameters (arguments) separated by spaces. All but the first will default usefully.

P1 is the search string (required). Leave all arguments blank and you get a brief reminder of how it works. You may enter more that one string separated by commas for a logical search that requires all strings to be in a source line, or separated by "|" for any of the strings being found in the source string. If a search string contains a space, enclose it in double quotes.

For a lot of my work, my default directory contains the files I want to search and all I have to do is type SEA 'string' and off it goes. If I need to be more specific with what files to search, I can provide an additional parameter.

P2 is what files to search and can contain as many file specification parts as you wich to provide. Those parts not given will be defaulted. You can get pretty fancy here. But, if you simply want to fill the place so you can pass the next argument, just type an asterisk.

P3 is the argument for /WINDOWS= which defaults to (0,0). You may type ',n', 'n,', 'n,m', 'n' or just ',' to hold the place. In the case of a single nummeric value, SEARCH will divide it into it's own idea of before and after.

P4 is a string of file types to exclude from the search in the form: '*.exe,*.mlb,*.obj,*.stb'. To hold the place in order to use the last parameter just enter any single character.

P5 Any character will cause the output to be placed in TEMPD:SEA.TXT and then typed out.

#____
MACRO32 & MACRO64 Structured Macros. Back in the day, programming languages all had the ubiquitous GOTO (or equivalent) statement. This led to code that was hard to understand and had numerous bugs that were difficult to impossible to find. Correcting them often led to more bugs. In the late seventies, structured code statements were introduced which led to much more understandable code that was infinitely easier to debug. I did this at the Ford lab, and later for my own company which used an early version of Disk Corp. DBL business language. I’m an assembly language programmer and have done this for MACRO 32 and MACRO64. You have an example in the .MAR] sub-directory, the .MLB files and the source for the macros in the .MLB] sub-directory.

This is controversial. I do not know why this is controversial as the code is usually as efficient as code without it and so very much more readable and reliable. Only rarely must code be as fast as possible. Most systems programs provided herein are run only occasionally. They need to be maintainable which means readable.

#____
CONNECTING to the system. - There are a variety of methods to connect to an OpenVMS system. If you are using AXP emulation software, you usually have the ability to connect to the operator's console and one or two COM communications ports. In some cases this is done with a browser running on a laptop or using a terminal emulator running on a laptop. Often this is sufficient for the hobbyist, but for real systems, addition connections are needed.

One way is the old fashioned terminal connector like a DecTerm that multiplexes many terminals connected over either twisted pair lines with DB25 connectors or Ethernet connectors. This is supported by some AXP emulators like AVT's. A less expensive and simpler way is to connect a laptop running a terminal emulator to a local router either with a radio or an Ethernet cable, the latter being more secure. This connects via TELNET emulation to the TCP/IP TELNET server service which is easily configured on VMS. This will handle large numbers of connections limited basically by the capacity of your VMS system and/or your router. If you use and Ethernet switch rather than a router, the connection will be more secure because a physical connection will be required.

If you need to allow connecting from outside over the internet, security becomes more difficult.

Router port forwarding to the standard TELNET port works - too well. I tried it.

within the hour the number of attempted logins increased to the point where the VMS system was brought to its knees. Not broken, but useless. Don''t do this.

My next attempt was to set port forwarding on the router to use a very non-standard port number. This actually worked pretty well. I left it like that for several months and don't believe there was a single attack. My good fortune.

A better way is to use Secure Shell or SSH-2. I set up port forwarding using a non-standard port, set up the VMS SSH-2 server to require both public/private key file pair authentication and VMS username/password authentication. I disseminated the private key with a passphrase attached. Thus, the login sequence requires an SSH connection to a known non-standard port, the user has his terminal emulator set to use SSH-2 with the properly formatted private key, the user is required to enter the passphrase associated with that private key and must then enter a valid VMS user name and password. You are not likely to have uninvited guests entering this way.

The setup for the PUTTY terminal emulator is in Appendix B and the way to handle the public/private key pair is described in Appendix C. Do not use the standard key pair generation technique offered in the TCP/IP configuration procedure. It generates useless files and sub-directories that only confuse the issue.

#____
This ends the ADVMGR manual documentation. I hope it suffices for your purposes. You can contact me if you need help or have questions.

If you are a programmer, you might be interested in reading ADVMGR_PROGRAMMER, a document full of all sorts of opinions, tips, tricks and other crap reflecting on fifty years of messing with computers as a research scientist, a businessman, a systems programmer and a provider of software.

Rick Marsh Ph.D.
Tulasi Software Systems
www.rickmarsh.com
rick@rickmarsh.com

Addendum A: The SYSUAF authorization data entry program.

Adding a new user is a two step process. Choose 1. Add a User. A listing of all the users will be shown to you. You will then be asked for a unique user name. the SYSUAF.LIS file will be searched to be sure it is unique. It will then ask for the UIC octal group number and octal member number. The UIC formed from these will be used to search SYSUAF.LIS to be sure it is unique. When all is well, it will add the user, update SYSUAF.LIS and throw you into the program we are discussing here. This program may be run from 3. Modify a User in which case it will ask for the user name to modify.

In adding a user, the fields you need to change are:

The default device (include the colon ":" at the end).
The default directory, [xxxx].
The default login script, LGICMD. Often LOGIN.COM on my system.
The password is blank, change it.
The user is added disabled. modify the flags vector accordingly.

You may wish to modify other fields at this time. For a new user, most fields are obtained from the user name DEFAULT which is used as a template. You may use this program to modify the DEFAULT user name as well.

DISCUSSION:

Please rely on the documentation in the System Managers Utilities Manual and HELP in AUTHORIZE for details regarding each field in SYSUAF.DAT (the authorization file) and the meaning of all the various values that make them up. This program makes changing many of them easier but does not help you understand what you are doing.

In order to maximize flexibility and longevity, this program reads in the flags values and privilege values from files: DATAD:FLGLST.DAT and DATAD:PRVLST.DAT. In turn, they are produced by COMD:CREFLGLST.COM AND COMD:CREPRVLST.COM respectively from SYS$LIBRARY:STARLET.REQ and should be run after each major upgrade of the OS.

The main screen displays a large amount of information gleaned from a SYSUAF.DAT record. The difference is that our display is a data entry screen with numbered fields you can select and change. Fields contain a variety of data types in a variety of formats; everything from simple numbers to huge bit vectors. There are also dates, names, a UIC field plus others. Numbers are decimal except for the UIC field which is octal. Alpha characters are forced by the data entry software to be uppercase.

On the primary window, field labels are color coded. Green is normal. Red displays information only and can't be changed. Light Blue means a different format with many possibilities which require a secondary screen for display and change. Orange means the DATE data represents a delta time (days) rather than a date. In the case of the orange UIC, it suggests you may not want to go there. When you add a new user, make sure the new UIC is unique. If it isn't, all sorts of problems will happen in the future with security. There may be legitimate reasons for changing the UIC on an existing user. We allow it and try to do the following. The program will create a sub-process that will attempt to add the new UIC to the rights list database. It will likely succeed with the new UIC, if it is unique and fail to add the username. The operation is logged and the log file printed on the screen so you can see any errors.

This will allow you to grant an identifier to the new UIC for security but will leave the old UIC in the rights list database which could lead to duplicate error messages in the future. The same process will change all the file ownership's to the new UIC.

A better way is to REMOVE the user, ADD the user with a unique UIC, GRANT the user the appropriate identifiers, use the menu selection to put the new ownership on all his files and then go into this modify program to finish setting any different parameters necessary. Cumbersome, but it avoids a raft of future problems.

On separate screens that represent data as bit vectors (flags and privileges) the color red indicates the flag is set or the privilege is granted. Primary days and Secondary days are listed separately. A day can be moved back and forth between the categories by selecting the day.

Access restrictions are by hour in a twenty-four hour vector. Each listed hour can be granted access (green) or denied access (red) by selecting its number.

Password change is done a bit differently from the standard way of doing things. See $SETUAI system service for details. If you enter a password for a user, we actually do the hash using existing salt and encryption algorithm fields in that record and then change the password change date. We do not allow you to change SALT and ENCRYPT in this program.

This operation covers the most often used fields in the SYSUAF record, but not all of them. For special cases, you can run AUTHORIZE from the same menu and do things by hand.