Chapter Eight

CUSTOMIZING YOUR WORK ENVIRONMENT

Introduction

The OpenVMS operating system offers a number of features that enable you to customize your work environment for convenience and efficiency.

- Commonly used files and directories can be represented by short names that are meaningful and logical to the user. These names are easier to type and remember than full file specifications.

- Symbols can be used as shortcuts for entering frequently used commands.

A special command file called LOGIN.COM can be created by the user to enable these features each time he logs in to the system.

This lesson introduces logical names and symbols and discusses keypad definitions and the simple login command feature.

Please be careful as you use the capabilities provided in this chapter. It is dangerous to redefine OpenVMS commands, either on the fly, or in a command file (such as LOGIN.COM). Always use the SuRPAS command file templates when creating SuRPAS command procedures.

Objectives

To customize the work environment to match specific needs, a user should be able to:

- Use logical names
- Use symbols
- Use keypad definitions
- Write a simple command procedure

Using Logical Names

Overview

A ***logical name*** is a character string that is used to represent other character strings called ***equivalence strings***. Once a logical name has been equated to a string, it can be used in place of the equivalence string. Equating a short logical name to a long file specification will reduce the typing effort and the likelihood of mistakes.

Using logical names to make programs independent of specific devices saves maintenance costs and makes programs portable to other OpenVMS systems. A program should be written to execute on several systems. However, the data may be located on a different disk than the program. The programmer may be unaware of the disk location of the data. Logical names for the data disk and directory can be defined by the program installation procedure.

This section discusses:

- Creating logical names
- Using logical names
- Determining the equivalence of a logical name
- Removing logical names
- Logical names created by the system

Creating Logical Names

Commands Used to Create Logical Names

There are two DCL commands that enable you to create logical names. Both the ASSIGN and DEFINE commands accomplish the same task while using different syntax. Both commands create logical names that are available to one user only. **When creating logical names in SuRPAS we only use the DEFINE command.**

ASSIGN command	
Syntax:	`ASSIGN` *`equivalence-string`*`[,...]`*`logical-name`*`[:]`
Example:	`$ ASSIGN DUA1:[9BR] BOB`
DEFINE command	
Syntax:	`DEFINE` *`logical-name`*`[:]` *`equivalence-string`*`[,...]`
Example:	`$ DEFINE BOB DUA1:[9BR]`

Rules for Creating Logical Names

A logical name can have a maximum of 255 characters and can contain alphanumeric characters, as well as the underscore (_), dollar sign ($), and hyphen (-).

The equivalence string can have a maximum of 255 characters as well. Like the logical, it can contain alphanumeric characters, the underscore (_), dollar sign ($), and hyphen (-). In addition, the equivalence string can contain colons, brackets and periods as required to be a part of a valid file specification.

Do not insert a colon at the end of a logical name. The colon is only used as a part of the equivalence string when specifying a device field within the file specification.

Logical Names In Equivalence Strings

The equivalence string for a logical name can contain another logical name.

In the following example the logical name DISK is defined, and then used in the equivalence string of the logical name REPORT.

```
$ DEFINE DISK DBA1
$ DEFINE REPORT DISK:[9RB.DATA]AUGUST.TXT
```

Duration Of A Logical Name

Logical names are not stored permanently in the system. When you create a logical name it is defined only for the life of the process. When you log out of the system the logical name is deassigned. If you want a logical name to be available every time you log in, you must create the logical name in a file to be executed each time you log in.

Logical Name Tables

The OpenVMS operating system maintains a number of logical name tables (LNM's). Each logical name table contains a set of logical names and their equivalent names. There is a logical name table created:

- when a process is created,
- when a job created,
- for each group of users or processes (using the UIC group field), and
- for the system.

A logical name table continues to exist as long as there is at least one object accessing the table. All logical name tables created in the OpenVMS operating system begin with the characters **LNM$**. In the case of a process logical name table, the table is deleted when the process terminates. The same is true for a job. If a job has created numerous processes, the job logical name table cannot be deleted until all processes in the job terminate.

SuRPAS-specific logical names are created at boot time in SYS$BOOT and are stored in the logical name table **LNM$FILE_DEV**.

Common Uses For Logical Names

A logical name is most commonly used in place of all or part of a file specification, frequently the directory portion. In the following example, the logical name CODE is equated to a directory specifcation, and then used in a command.

```
$ DEFINE  CODE      [9RB.CODE]
$ DIRECTORY  CODE

Directory  MISC$DISK:[9RB.CODE]
     •
     •
     •
```

Example 8-1 - Using a Logical Name for a Directory Specification

When used to refer to a file specification, a logical name may represent the entire file specification or just its leftmost portion. In the following example, the logical name AUGUST is equated to the complete file specification.

```
$ DEFINE  AUGUST    MISC$DISK:[9RB.WORKSHEET]AUGUST_RESULTS.DAT
$ DIRECTORY    AUGUST

Directory  MISC$DISK:[9RB.WORKSHEET]

AUGUST_RESULTS.DAT;1

Total of 1 file.
$
```

Example 8-2 - Using a Logical Name for a Complete File Specification

In the following example a logical name is used to replace the left part of a file specification. The logical name CODE is equated to the disk and directory in which a particular data file is located. The logical name is used as the left part of the file specifcation and separated from the file name by a colon (:).

```
$ DEFINE  WS   MISC$DISK:[9RB.WORKSHEET]
$ DIRECTORY  WS:AUGUST_RESULTS.DAT

Directory  MISC$DISK:[9RB.WORKSHEET]

AUGUST_RESULTS.DAT;1

Total of 1 file.
$
```

Example 8-3 - Using a Logical Name in the Left Part of the File Specification

A logical name can NOT be used to replace the right-hand portion of a file specification.

```
$ DEFINE  JULY      JULY_RESULTS.DAT
$ DIRECTORY    [9RB.WORKSHEET]JULY
%DIRECT-W-NOFILES, no files found
$ DIRECTORY  JULY

Directory  MISC$DISK:[9RB.WORKSHEET]

JULY_RESULTS.DAT;1

Total of 1 file.
$
```

Example 8-4 - Using a Logical Name in the Right Portion of a File Specification

Translating Logical Names

When the equivalence string for a logical name is a logical name itself, the translation process can be done in two ways.

- Translate only the first logical name, using the SHOW TRANSLATION command.
- Translate the logical name and a logical name found in the equivalence string, repeating the process for any further logical names found. This is called ***iterative translation,*** and the SHOW LOGICAL command is used.

The SHOW TRANSLATION Command

The SHOW TRANSLATION command is used to display the equivalence string of a logical name. Any additional logical names contained in the equivalence string will be displayed, but not translated. The syntax for the SHOW TRANSLATION command is as follows:

```
        SHOW TRANSLATION logical-name
```

In the following example, the logical name PAYROLL is assigned to another logical name in the equivalence string (DISK1). The SHOW TRANSLATION command displays the translation of the first logical name only.

```
$ DEFINE      DISK1          DJA0:
$ DEFINE:     PAYROLL        DISK1
$ SHOW TRANSLATION           PAYROLL
  PAYROLL = "DISK1:"         (LNM$PROCESS_TABLE)
$
```

Example 8-5 - The SHOW TRANSLATION Command

The SHOW LOGICAL Command

The SHOW LOGICAL command iteratively translates the logical names until all logical names are resolved (up to 10 levels). The syntax for the SHOW LOGICAL command is as follows:

```
SHOW LOGICAL logical-name
```

In the following example, the logical name DISK1 is used in the equivalence string for the logical name PAYROLL. The SHOW LOGICAL command is used to display the iterative translation.

```
$ DEFINE     DISK1          DJA0:
$ DEFINE     PAYROLL        DISK1:
$ SHOW LOGICAL  PAYROLL
        "PAYROLL" = "DISK1"         (LNM$PROCESS_TABLE)
1       "DISK1" = "DJA0:"           (LNM$PROCESS_TABLE)
$
```

Example 8-6 - The SHOW LOGICAL Command

Removing Logical Names

The DEASSIGN Command

Use the DEASSIGN command to cancel logical name assignments made with either the ASSIGN or DEFINE commands. The syntax for the DEASSIGN command is as follows:

```
        DEASSIGN   logical-name
```

For example, the following command removes the logical name PAYROLL.

```
        $ DEASSIGN PAYROLL
```

Delete All Process Logical Names

Use the /ALL qualifier on the DEASSIGN command to cancel all of your process logical names. The syntax for the DEASSIGN/ALL command is as follows:

```
        DEASSIGN/ALL
```

Logical Names Created by the System

The OpenVMS operating System creates several logical names for your process. Two of these logical names are SYS$LOGIN and SYS$LOGIN_DEVICE.

SYS$LOGIN

When you log in, the OpenVMS operating system equates the logical name SYS$LOGIN to your initial default device and directory. This may be thought of as your home directory.

You can use the logical name SYS$LOGIN to return to your home device and directory after you have changed your default directory to another device and/or directory.

```
$ SET DEFAULT SYS$LOGIN
$
```

Example 8-7 - The SYS$LOGIN Logical Name

SYS$LOGIN_DEVICE

The OpenVMS operating system equates the logical name SYS$LOGIN_DEVICE to the disk containing your login directory.

Use the SHOW DEVICE command with the logical name SYS$LOGIN_DEVICE to obtain information about your disk.

```
$ SHOW DEVICE SYS$LOGIN_DEVICE

Device               Device    Error  Volume    Free  Trans  Mnt
 Name:               Status    Count   Label  Blocks  Count  Cnt
$1$DUA1: (DONALD)    Mounted     0     USER1  167538    2     12
$
```

Example 8-8 - The SYS$LOGIN_DEVICE Logical Name

Using Symbols

Overview

A symbol is a name that represents a numeric, character, or logical value. When you use a symbol in a DCL command line, DCL uses the value you assign to that symbol. By defining a symbol as a command line, you can execute the command by typing only the symbol name.

This section discusses:

- An introduction to symbols
- Creating symbols
- Displaying symbols
- Deleting symbols

Introduction To Symbols

Symbols are useful shortcuts for entering frequently used DCL commands, and they can represent data in command procedures.

A symbol can be used:

- As a synonym for an entire DCL command.
- As a variable within a DCL command procedure.
- As a parameter (argument) passed to a command procedure.

The value you assign to a symbol can be made available to the command interpreter either locally or globally.

Local symbols are usually used as variables within command procedures. A local symbol exists:

- During the execution of the command procedure that created it.
- In command procedures invoked by the procedure that created it.

When the procedure that has created a local symbol exits, the local symbol ceases to exist.

A **global symbol** applies to all command levels, even after the command procedure that created it has exited.

Duration Of Symbols

Symbols are not stored permanently in the system. When you create a symbol, it is defined only for the life of the process. When you log out of the system, the symbol is deassigned. If you want a symbol to be available every time you log in, you must create the symbol in a file to be executed each time you log in.

Data Types For Symbols

A symbol can represent:

- A **character string**, containing any characters that can be printed. You can include spaces in the string and preserve the use of upper- or lower-case letters.

- An **integer number**, positive or negative. To specify a number in a radix other than decimal (base 10), precede the number (but not the sign) with **%X** for hexadecimal values (e.g. %X27) and **%O** for octal values (e.g. %O45).

Creating Symbols

Equating Symbols To Character Strings

Use a colon (:) and an equal sign (=) to assign character string values to a symbol.

- Use one colon and one equal sign to create a local symbol:

  local-symbol* := *string

```
$ DISK := MISC$DISK
$ BADGE_NUMBER := 125
```

- Use one colon and two equal signs to create a global symbol:

  global-symbol* :== *string

```
$ GLOBAL_DISK :== MISC$DISK
```

To preserve case and spaces in a string:

```
$ ERROR_MESSAGE_1 = “File NOT found.”
$ GLOBAL_ERROR_MESSAGE == “The file was NOT
found.”
```

When using the “**:=**” to equate symbols to character strings think of the colon as replacing a pair of double quotes (“) around the string!!

Equating Symbols to Integers

Use the equal sign (=) to assign an integer value to a symbol.

- Use one equal sign to create a local symbol:

  local-symbol = integer-value

```
$ COUNT = 1
```

- Use two equal signs to create a global symbol:

  global-symbol == integer-value

```
$ NEWCOUNT == 100
```

Replacing a Command with a Symbol

Equating a symbol to a command line lets you execute the command by typing the symbol name only.

The following examples illustrate two ways to use a symbol to redefine the top level directory as the default directory when subdirectories are used.

```
$ HOME :== SET DEFAULT SYS$LOGIN
$ HOME == “SET DEFAULT SYS$LOGIN”
```

In order to send print jobs to a printer called TEXT_PRINTER_2, the following symbol will allow you to send a print job by typing PRN2 followed by the file name.

```
$ PRN2 == “PRINT/QUEUE=TEXT_PRINTER_2”
```

Once this symbol has been defined, the README.TXT file can be printed by typing the following:

```
$ PRN2 README.TXT
```

You can define the symbol UP to set your default directory to the preceding level directory by entering:

```
$ UP == “SET DEFAULT [-]”
```

Creating Symbols That Can be Abbreviated

Symbols defined with an asterisk (*) can be abbreviated. Place the asterisk after the letter that makes the abbreviation unique to the operating system. This allows you to type a portion of the symbol name as long as it is at least the minimum abbreviation defined. The following is an example of a global symbol that can be abbreviated:

```
$ CL*EANUP == “PURGE/LOG”
```

After this symbol has been created, any of the following commands will cause the PURGE/LOG command to execute:

```
$ CL
$ CLE
$ CLEA
$ CLEAN
$ CLEANU
$ CLEANUP
```

Displaying Symbols

Displaying The Equivalence String

Use the SHOW SYMBOL command to display the equivalence string for a symbol with the following syntax:

```
SHOW SYMBOL [/GLOBAL] [/ALL]  symbol-name
```

Sample SHOW SYMBOL Commands

If you want to display...	**Use this command:**
A local symbol X	`$ SHOW SYMBOL X`
All local symbols	`$ SHOW SYMBOL/ALL`
A global symbol X	`$ SHOW SYMBOL/GLOBAL X`
All global symbols	`$ SHOW SYMBOL/GLOBAL/ALL`

Deleting Symbols

Use the DELETE/SYMBOL command to cancel a symbol definition with the following syntax:

```
DELETE/SYMBOL [/GLOBAL] symbol-name
```

Sample DELETE/SYMBOL Commands

If you want to delete...	Use this command:
A single local symbol	`$ DELETE/SYMBOL X`
All local symbols	`$ DELETE/SYMBOL/ALL`
A single global symbol	`$ DELETE/SYMBOL/GLOBAL GX`
All global symbols	`$ DELETE/SYMBOL/GLOBAL/ALL`

Writing Command Procedures

Overview

A file can be created which contains frequently executed sequences of DCL commands. This file, called a **command procedure**, saves the user from repeatedly typing the command sequences, and can be executed interactively or in batch mode.

This section discusses:

- Developing command procedures.
- Executing command procedures.
- Writing a login command procedure.

Developing Command Procedures

Steps for Developing Command Procedures

Follow these steps to develop a command procedure:

1. **Design the command procedure**.
    a. Determine what tasks the procedure should perform.
    b. Decide what results the procedure should produce.

2. **Create the command procedure file.**
    a. Use the text editor of your choice (EVE, EDT, or AMS).
    b. Assign the command procedure the file type COM.

3. **Execute and test the command procedure.**

4. **Modify and retest the command procedure,** if necessary, by repeating steps 2 and 3.

5. **Add comments to simplify maintenance of the command procedure.** Comments may include:
    - The name of the procedure, the author, and the last revision date.
    - A description of what the command procedure does, including kind(s) of input expected.
    - Descriptions of specific commands and groups of commands included in the command procedure.

Rules For Developing Command Procedures

When writing a command procedure, follow these rules:

- Start each DCL command on a new line.
- Precede each DCL command with a dollar sign ($).
- Use a hyphen (-) to continue a command on the next line. Do not put a dollar sign ($) on the continuation line (s).
- Precede comments with an exclamation point (!).

Guidelines For Developing Command Procedures

In addition to the basic rules for construction of a command procedure, it is essential to use consistent format; this ensures command procedures that are easy to read, test, and maintain.

- In order to execute a command procedure, it is necessary to specify a file type. For consistency, assign the operating system default file type (COM) to all command procedures.
- Always use full DCL commands, not abbreviations. While full commands will not change in future versions of the OpenVMS operating system, abbreviated commands may not remain unique as more commands are added to DCL.
- For readability, use comment lines (containing only the dollar sign ($) and exclamation point (!) in the first two columns) to separate groups of commands.
- End your command procedure with the EXIT command to clearly indicate its end point. This will help with maintenance, especially when numerous command procedures are being executed from within a command procedure.

The following example shows a simple command procedure that is intended to execute when the user modifies a FORTRAN source code file. It recompiles and relinks a program named MARKET_CALC.

```
$!   BUILD_MARKET_CALC.COM - Compile and Link the MARKET_CALC
$!                           program
$!   Author:                 Bradley Lord
$!   Last Revised:           23 OCT 2001
$!
$    FORTRAN/LIST/NODEBUG  MARKET_CALC  ! Compile source code
$!
$    LINK/MAP/NODEBUG     MARKET_CALC   ! Link the object code
$!
$    EXIT
```

Example 8-9 - A Simple Command Procedure

Executing A Command Procedure

DCL command procedures are executed by the DCL Command Language Interpreter (CLI). You cannot compile a DCL command procedure.

Command procedures can be executed interactively from the DCL command level. When a command procedure executes interactively in this manner, output is displayed on your terminal. Your terminal will not respond to any commands while the command procedure is executing.

To execute a command procedure interactively, type the @ (pronounced 'at') symbol followed by the file specification of the command procedure.

```
@command-procedure-name.COM
```

The following example demonstrates how to execute the BUILD_MARKET_CALC.COM command procedure:

```
$  @BUILD_MARKET_CALC.COM
$
```

Example 8-10 - Executing A Command Procedure

If no file type is specified, the system uses the default file type (COM).

Testing Your Command Procedure

Use the SET VERIFY command to test and debug your command procedure; add the command as the first line. This will cause the system to display each line of your command procedure before executing it. To turn this feature off, add the SET NOVERIFY command to the end of the procedure (just before the EXIT command).

When you finish testing your command procedure you can remove the SET VERIFY and SET NOVERIFY commands or comment them out by adding an exclamation point (!) to the beginning of each of the two command lines.

You can also enter the SET VERIFY and SET NOVERIFY commands interactively. Make sure to use the commands in pairs when adding them to a command procedure. If you leave out the SETNOVERIFY command at the end of the command procedure, the VERIFY will remain in effect after you leave the command procedure.

The following example illustrates the use of the SETVERIFY and SET NOVERIFY commands.

```
$ SET VERIFY
$ @BUILD_MARKET_CALC
$!   BUILD_MARKET_CALC.COM - Compile and Link the MARKET_CALC
$!                            program
$!   Author:                  Bradley Lord
$!   Last Revised:            23 OCT 2001
$!
$    FORTRAN/LIST/NODEBUG MARKET_CALC ! Compile source code
$!
$    LINK/MAP/NODEBUG    MARKET_CALC  ! Link the object code
$!
$    EXIT
$ SET NOVERIFY
$
```

Example 8-11 - Using SET VERIFY and SET NOVERIFY

Notes: Using SET VERIFY and SET NOVERIFY

- Enter the SET VERIFY command to cause the system to display each line.
- Execute the command procedure (BUILD_MARKET_CALC).
- Note that each line of the command procedure is displayed before it is executed. Any output from the command procedure will be displayed along with any comment lines.
- After the command procedure exits, you can enter the SET NOVERIFY command to turn off verification.

Halting Execution

You can halt the execution of an interactive command procedure by typing the Ctrl/Y key or the Ctrl/C key. Because the Ctrl/C key may cause some programs to take special action, it is recommended that the Ctrl/Y key be used.

If you press the Ctrl/Y key and halt execution, you can use the CONTINUE command to resume (as long as no intervening DCL commands have been entered).

Executing A Command Procedure As A Batch Job

You can execute a command procedure as a batch job by using the SUBMIT/QUEUE command to place the job in the batch queue. This allows you to use your terminal for other work while the command procedure is executing.

The following example illustrates the execution of the BUILD_MARKET_CALC command procedure as a batch job.

```
$  SUBMIT/QUEUE  BUILD_MARKET_CALC
   Job  BUILD_MARKET_CALC  (queue  SYS$BATCH,  entry  416)  pending
$
```

Example 8-12 - Executing A Command Procedure in Batch Mode

Aborting a Batch Job

Once a batch job has been queued for processing using the SUBMIT/QUEUE command, the user that issued the SUBMIT command can abort the job by issuing a STOP/QUEUE/ENTRY command. When issuing this command, the user must supply the entry number of the job. This entry number was provided by OpenVMS when the SUBMIT command was processed. The syntax of this command is:

```
$  SUBMIT/QUEUE/ENTRY=entry-number    [queue-name]
```

Writing A Login Command Procedure

LOGIN.COM

You can create a command procedure to execute the same commands each time you log in to your system. Name the file LOGIN.COM and place it in the top level of your default user directory (unless your system manager instructs you otherwise).

The file should contain the commands to set up your OpenVMS environment, including the following, which will be available to your process each time you log in:

- symbols
- logical names
- key definitions

Bypassing LOGIN.COM

If you modify your LOGIN.COM, be sure to test it before you log out. If your LOGIN.COM contains errors that prevent you from logging in, bypass it by adding the /NOCOMMAND qualifier to your user name when you attempt to log in:

```
Username:    9RB/NOCOMMAND
```

Most first login command procedures are created by copying someone else’s and making minor changes. As the user becomes more comfortable and skilled in DCL, the login procedure is modified to suit his specific needs.

The sample login command procedure on the next page illustrates some of the rules and guidelines that have been discussed in this section; it also contains some additional advanced information that has not yet been discussed. It can be used as a template for your own LOGIN.COM file.

```
$ set noverify
$ DEFINE/PROC/EXEC MAGIC_KEY 96 ! MAKE THE MAGIC-WINDOWS KEY A TILDE CHAR.
$ !—————————————————————————————-——————————
$ define/job eve$init sys$login:eveinit.eve
$ define tpu$section FAL$COMMON:AM$EDIT.INI
$ !
$ ! DEFINEs for AMS extensions to TPU editor
$ define tpu$command FAL$COMMON:AM$CFG_FAL.TPU
$ define tpu$am_search_dir [],fal$sub:,fal$main:,fal$com:,fal$scr:,fal$files
$ AMCFG_COMPILE_CMD_FOR :== “FOR”
$!
$ IF F$MODE() .NES. “INTERACTIVE” THEN $GOTO EXIT
$ DEFINE SUB            FAL$SUB
$ DEFINE MAIN           FAL$MAIN
$ DEFINE PRG            FAL$PRGLIB
$ DEFINE FILES          FAL$FILES
$ !
$ CLS           :== SET TERM/WIDTH=80
$ TI            :== SHOW TIME
$ FOR           :== “$FAL$FORTRAN/OBJ=*.OBJ”
$ LINK          :== LINK/EXEC=*.EXE
$ ED*IT         :== EDIT/TPU
$ EDT           :== EDIT/EDT
$ DIR           :== DIR/DATE=MOD/SIZE=ALL/WIDTH=FILENAME=35
$ MD            :== CREATE/DIR
$ NORM*AL       :== SET TERM/WIDTH=80
$ Q             :== SHOW QUE HPJET4
$ QF            :== SHOW QUEUE/ALL FTK$PRINT
$ REN*AME       :== RENAME/LOG
$ SUB*MIT       :== SUBMIT/NOPRINT/KEEP/NOTIFY
$ TIM*E         :== SHOW TIME
$ WB            :== RUN FAL$PRGLIB:WBTOPMEN
$ SP            :== PRINT/NOTIFY/FLAG/QUEUE=SYS$PRINT
$ !
$ PRN           :== PRINT/NOTIFY/NOFLAG/QUEUE=HPJET4/FORM=HPORT132/HEADER
$ PRNW          :== PRINT/NOTIFY/NOFLAG/QUEUE=HPJET4/FORM=DIFFS/HEADER
$ PUR*GE        :== PURGE/LOG
$ DEL*ETE       :== DELETE/LOG
$ DAT*E         :== SHOW TIME
$ LNM           :== @fal$common:PROMPT.COM ! should be fal$common
$ LO            :== EOJ
$ define/job ftk$print hpjet4
$ !
$ set terminal/line_edit/insert
$ !
$ ! Start MAGICwindows (must be last item in LOGIN.COM)
$ !   INQUIRE YN “>>> Do you want to run MAGICwindows [Y]”
$ !   IF YN .EQS. “” .OR. YN .EQS. “Y” THEN MAGIC
$ ! —————————————————
$ MAGIC
$ EXIT:
$   EXIT
```

Example 8-13 - Abbreviated Training LOGIN.COM Command Procedure

Common Problems in Login Command Procedures

The following is a list of common problems and some suggested solutions.

- **You are immediately logged out.**
  Check to see if LOGIN.COM contains a LOGOUT command. If so, log in without allowing the LOGIN.COM procedure to execute, and then edit the LOGOUT command out of your LOGIN.COM file.

- **None of your customizations are in effect.**
  These features will not be available until LOGIN.COM has executed. LOGIN.COM may be executed by typing the DCL command @LOGIN (or you may simply log out and log back in).

  If the problem persists, force some command line displays within the command procedure to determine if the LOGIN.COM procedure is running. This can easily be done by using the SET VERIFY command within the LOGIN.COM file or including commands that display output on the screen when executed like the SHOW TIME or SHOW USER DCL commands.

- **Some symbols do not work.**
  If your symbols will be used outside of LOGIN.COM, they must be defined as global (rather than local). The use of *two* equal signs (==) is necessary to force a global definition.

```
$ HOME == “SET DEFAULT SYS$LOGIN”
```

- **Some key definitions do not work.**
  You must execute one of the following commands to enable your keypad to accept definitions:

```
$ SET TERMINAL/NONUMERIC
$ SET TERMINAL/APPLICATION_KEYPAD
```

- **Some DCL commands do not work as they should.**
  Avoid creating a symbol with the same name as a DCL command; the symbol will always override the DCL command. In the following example, a global symbol is created with the same name as the DELETE command. The user will no longer have the ability to delete files; instead, he will be logged out of the system when the DELETE command is issued.

  ```
  $ DEL*ETE == “LOGOUT”
  ```

- **Some features work and others do not.**
  If there is an error in your LOGIN.COM file, some commands will not be executed. Enter the SET VERIFY command; then test your LOGIN.COM procedure interactively and watch as each command executes. Notice where the execution halts. Check for error messages on the screen to help you debug the problem.

- **You receive “directory not found” error messages.**
  You copied someone else’s LOGIN.COM file and did not replace their directory specification with your own. Edit the LOGIN.COM file and make sure that you have specified the correct directories. When possible, define symbols to represent your directory specifications; the change can be made once in the procedure rather than numerous times. This will help in avoiding missed changes.

The PFPC Manufacturing Group LOGIN.COM

The LOGIN.COM found in your default student directory and a good model for your own LOGIN.COM is found in the following examples. Each section manages a different goal for the LOGIN.COM file.

The first section initializes your environment when you log into OpenVMS as a user. The screen attributes are set, the tilda key (~) is defined for MAGICwindows and the EVE and AMS editing environment is initialized.

```
$ set noverify
$ set term /vt100
$ SET CONTROL = (T,Y)                    ! allow control-T and control-Y for programmers
$ DEFINE/PROC/EXEC MAGIC_KEY 96   ! MAKE THE MAGIC-WINDOWS KEY A TILDA
$ !
$ !-----------------------------------------------------------------------
$ define/job eve$init sys$login:eveinit.eve
$ define tpu$section FAL$COMMON:AM$EDIT.INI
$ !
$ ! DEFINEs for AMS extensions to TPU editor
$ !
$ define tpu$command FAL$COMMON:AM$CFG_FAL.TPU
$ define tpu$am_search_dir [],fal$sub:,fal$main:,fal$com:,fal$scr:,fal$files
$ AMCFG_COMPILE_CMD_FOR :== "FOR"
$!
```

Example 8-14 - Initial section of the PFPC LOGIN.COM

The next section begins by limiting execution of the rest of the commands to interactive mode, by testing the current mode. This section then defines shortcuts to the more popular SuRPAS directories and sub-directories.

```
$ IF F$MODE() .NES. "INTERACTIVE" THEN $GOTO EXIT
$!
$ DEFINE SUB        FAL$SUB
$ DEFINE MAIN       FAL$MAIN
$ DEFINE REP        FAL$REPORT
$ DEFINE RPT        FAL$REPORT
$ DEFINE PRG        FAL$PRGLIB
$ DEFINE FILES      FAL$FILES
$ DEFINE SCR        FAL$SCR
$ DEFINE TOOL       FAL$TOOL
$ !
```

Example 8-15 - Interactive mode test in the PFPC LOGIN.COM

This section of LOGIN.COM defines global symbols to be used during the interactive user session. In most cases the symbols are shortcuts for much longer DCL commands (e.g. TI:==SHOW TIME), in other cases the DCL command remains the same but there are additional, non-default attributes added to the command (e.g. DI:==DIR/DATE=MODIFIED/SIZE=ALL). There are even symbols defined to look like commands found in other operating systems, such as the DOS MAKE DIRECTORY (MD) command which replaces the CREATE/DIR DCL command.

```
$ FALLINK    :== @FAL$PRGLIB:FALLINK
$ SD         :== SET DEF
$ SHD        :== SHOW DEFAULT
$ RE         :== RECALL
$ CLS        :== SET TERM/WIDTH=80
$ WIDE       :== SET TERM/WIDTH=132
$ COPY       :== COPY/LOG
$ TI         :== SHOW TIME
$ DI         :== DIR/DATE=MODIFIED/SIZE=ALL
$ DIRF       :== DIR/SIZE/DATE *.FOR;
$ FOR        :== "$FAL$FORTRAN/OBJ=*.OBJ"
$ DFOR       :== "$FAL$FORTRAN/DEBUG/NOOPT/OBJ=*.OBJ"
$ LINK       :== LINK/EXEC=*.EXE
$ DLINK      :== LINK/DEBUG/EXEC=*.EXE
$ ED*IT      :== EDIT/TPU
$ EDT        :== EDIT/EDT
$ DIR        :== DIR/DATE=MOD/SIZE=ALL/WIDTH=FILENAME=35
$ DIRW       :== DIR/BRIEF
$ MD         :== CREATE/DIR
$ NORM*AL    :== SET TERM/WIDTH=80
$ Q          :== SHOW QUE HPJET4
$ QF         :== SHOW QUEUE/ALL FTK$PRINT
$ REN*AME    :== RENAME/LOG
$ SDA        :== ANALYZE/SYSTEM
$ SPY        :== RUN FAL$PRGLIB:SPY.EXE
$ SYS        :== SHOW SYS
$ SUB*MIT    :== SUBMIT/NOPRINT/KEEP/NOTIFY
$ TIM*E      :== SHOW TIME
$ USERS      :== SHOW USERS/NODE='FAL_NODE'/FULL
$ WB         :== RUN FAL$PRGLIB:WBTOPMEN
$ JBXLIST    :== RUN FAL$TOOL:JBXLIST.EXE
$ TPU        :== EDIT/TPU
$ SP         :== PRINT/NOTIFY/FLAG/QUEUE=SYS$PRINT
$ !
```

Example 8-16 - Global symbols section of the PFPC LOGIN.COM

This last section of LOGIN.COM completes the symbolic definition section, sets the terminal editing mode into insert mode, then exits.

```
$ PRN        :== PRINT/NOTIFY/NOFLAG/QUEUE=HPJET4/FORM=HPORT132/HEADER
$ PRNW       :== PRINT/NOTIFY/NOFLAG/QUEUE=HPJET4/FORM=DIFFS/HEADER
$ PR1        :== PRINT/NOTIFY/NOFLAG/QUEUE=HPJET1/FORM=HPORT132/HEADER
$ PR4        :== PRINT/NOTIFY/NOFLAG/QUEUE=HPJET4/FORM=HPORT132/HEADER
$ PR5        :== PRINT/NOTIFY/NOFLAG/QUEUE=HPJET5/FORM=HPORT132/HEADER
$ PR9        :== PRINT/NOTIFY/NOFLAG/QUEUE=HPJET9/FORM=HPORT132/HEADER
$ LAND       :== PRINT/NOTIFY/NOFLAG/QUEUE=HPJET4/FORM=HLAND/HEADER
$ EMAIL      :== MAIL/EDIT=(SEND)
$ DETAIL     :== DUMP/HEADER/FILE_HEADER/FORMATTED/BLOCKS=COUNT:0
$ SCREEN  :== RUN FAL$TOOL:SCREEN
$ FORGEN  :== RUN FAL$TOOL:FORGEN
$ RN         :== RUN/NODEBUG
$ USAGE    :== MONITOR PROCESSES/TOPCPU
$ PUR*GE   :== PURGE/LOG
$ DEL*ETE :== DELETE/LOG
$ DAT*E     :== SHOW TIME
$ LNM        :== @fal$common:PROMPT.COM ! should be fal$common
$ LO          :== EOJ
$ GETPAR  :== @lib/ext='P1'.PAR FAL$LIBRARY:Params.tlb/OUT='P1'.PAR
$ GETCFD  :== @lib/ext='P1'.CFD FAL$LIBRARY:FILES.tlb/OUT='P1'.CFD
$ GETCRD :== @lib/ext='P1'.CRD FAL$LIBRARY:FILES.tlb/OUT='P1'.CRD
$ LX          :== @DEV$DISK:[GWV.UTL]LIBREXT
$ FTKMARK :== Y
$ TLA         :== @USERUTIL$COM:TLA
$ define/job ftk$print hpjet4
$ !
$ set terminal/line_edit/insert
$ !
$ ! Start MAGICwindows (must be last item in LOGIN.COM)
$ !   INQUIRE YN ">>> Do you want to run MAGICwindows [Y]"
$ !   IF YN .EQS. "" .OR. YN .EQS. "Y" THEN MAGIC
$ ! ---------------------------------
$ MAGIC
$ EXIT:
$   EXIT
```

Example 8-17 - Final section of the PFPC LOGIN.COM

Summary

Concepts

Using Logical Names

- Use the DEFINE command to create a logical name.
- Use a logical name in place of all or part of a file specification.
- Use the SHOW LOGICAL or the SHOW TRANSLATION command to determine the equivalence of a logical name.
- The system creates logical names for your process when you log in.
- Use the DEASSIGN command to delete a logical name.

Using Symbols

- Create symbols to use as synonyms for frequently used commands.
- Use one or two equal signs (=) to equate a symbol name with a value, depending on whether the symbol is global or local.
- Use the SHOW SYMBOL command to display the contents of a symbol.
- Use the DELETE SYMBOL command to delete a symbol.

Writing Command Procedures

- Consistent format makes command procedures easy to read, test, and maintain.
- Use the @ (pronounced AT) command to execute a command procedure interactively.
- Use the SUBMIT command to execute a command procedure in batch.
- Use a LOGIN.COM command procedure to set up logical names, symbols, keypad definitions, and other characteristics of your terminal session each time you log in.

Commands

Using Logical Names

ASSIGN
Define a logical name.
DEFINE
Define a logical name.
SHOW DEVICE
Display information about a disk or other device.
SHOW LOGICAL
Display a logical name iteratively.
SHOW TRANSLATION
Display a logical name.
DEASSIGN
Remove a logical name.

Using Symbols

SHOW SYMBOL
Display a symbol.
DELETE/SYMBOL
Cancel a symbol definition.

Writing Command Procedures

@file-specification
Execute a command procedure interactively.
SET [NO]VERIFY
Toggle the display of commands in a command procedure.
SUBMIT *file-specification*
Execute a command procedure in batch mode.