Chapter Eleven

INPUT AND OUTPUT IN A COMMAND PROCEDURE

Introduction

Once data is organized into files, a common task is to write a procedure that will find a file and use the information in it. In addition, you may want the command procedure to ask the user for the name of the file to be accessed or to request a value indicating one of several possible tasks to be performed.

These and many other tasks can be performed in a single command procedure if it is written with the flexibility of querying the user about how the procedure is to run.

This lesson discusses how command procedures handle input and output (I/O). Specifically, we will look at terminal I/O (information typed to and taken from the screen) and file I/O (information written to or read from a file).

Objectives

Upon completion of this lesson, students will have the ability to provide data to and receive data from command procedures. Programmers will be able to write procedures that perform the following tasks:

- Display information at the user's terminal
- Request information from the user
- Place information into files
- Obtain information from files
- Redefine system logical names to redirect input and output

Chapter Terms

The topics discussed in this lesson assume that the student is familiar with the following terms.

I/O	Input and Output functions.
Lexical Function	A DCL command language construct that returns information about an item or list of items.
Looping	A process that repeats a set routine over and over until a particular requirement is met.
Symbol	An entity that, when defined, represents a function such as a command string or a directory name.
System Logical Name	A system-specified name for a particular location, such as where the system displays and reads information from the terminal.

Logical Names the System Creates for You

Logical names are important because:

- They are used in command procedures.
- They can be used to capture output from commands.

Process-permanent logical names are created by DCL when you log in, and they remain defined for the life of your process.

- You cannot delete these logical names.
- You can redefine a logical name by specifying the same name in a DEFINE or ASSIGN command. If the name is deassigned, the process-permanent name is re-established.

The table below lists the most popular process-permanent logical names provided by the operating system for your process.

SYS$INPUT	The default device or file from which DCL reads input
SYS$OUTPUT	The default file (usually your terminal) to which DCL writes output
SYS$ERROR	The default device or file to which DCL writes error messages generated by warnings, errors, and severe errors
SYS$COMMAND	The initial file (usually your terminal) from which DCL reads input - DCL uses this logical name to remember your initial input stream

Table 12-1 - Process-Permanent Logical Names

Use the SHOW LOGICAL/PROCESS command to display the equivalence of these process-permanent logical names.

```
$ SHOW LOGICAL/PROCESS

(LNM$PROCESS_TABLE)

   "SYS$COMMAND" = "_ALPHA1$VTA2347:"
   "SYS$DISK" = "MISC$DISK:"
   "SYS$ERROR" [super] = "_SYSLOGIN:FALUSER.ERR"
   "SYS$ERROR" [exec] = "_ALPHA1$VTA2347:"
   "SYS$INPUT" = "_ALPHA1$VTA2347:"
   "SYS$OUTPUT" [super] = "_ALPHA1$VTA2347:"
   "SYS$OUTPUT" [exec] = "_ALPHA1$VTA2347:"
   "TT" = "VTA2347:"
$
```

Example 11-1 - Process Logical Names

Using a Process-Permanent Logical Name to Redirect Output

You may want to redirect output to a file to keep:

- A directory listing for future reference
- A record of output from a program
- A list of your mail messages
- A copy of an error message (to mail to the system manager)

It is sometimes helpful to capture output for later reference. You can redirect terminal output to a file by using the ASSIGN or DEFINE command to give a new value to the logical name associated with your terminal. *Remember the when building command procedures in SuRPAS, the DEFINE command is preferred rather than the ASSIGN command.*

The following is the syntax for the ASSIGN and DEFINE commands when causing redirection:

```
$ ASSIGN output-file    SYS$OUTPUT
$ DEFINE SYS$OUTPUT     output-file
```

The sample commands below direct your output to the file STATS.DAT:

```
$ ASSIGN    STATS.DAT   SYS$OUTPUT
$ DEFINE    SYS$OUTPUT  STATS.DAT
```

This assignment will remain in place until you enter the DEASSIGN command or log out of the system.

You can make a temporary assignment by adding the /USER qualifier to the command. This assignment will last only during the execution of the next executable image (program).

In the example that follows we see how to direct output to a file. The file can later be edited, sent to a printer, mailed to another user, and so on.

```
$ SHOW PROCESS

10-DEC-2001 09:26:18.17 User: LEVINEJ   Process ID:    20200250
                        Node: TIGGER    Process name: “LEVINEJ”
Terminal:            FTA4:    (SUPER::LEVINEJ)
User Identifier:     [LEVINEJ]
Baes priority:       4
Default file spec: DISK$USER:[LEVINEJ]

Devices allocated: TIGGER$FTA4:
$
$ ASSIGN PROCESS.TXT SYS$OUTPUT
$ SHOW PROCESS
$ DEASSIGN SYS$OUTPUT
$TYPE PROCESS.TXT

10-DEC-2001 09:26:49.87 User: LEVINEJ   Process ID:    20200250
                        Node: TIGGER    Process name: “LEVINEJ”
Terminal:            FTA4:    (SUPER::LEVINEJ)
User Identifier:     [LEVINEJ]
Baes priority:       4
Default file spec: DISK$USER:[LEVINEJ]

Devices allocated: TIGGER$FTA4:
$
```

Example 11-2 - Redirecting Output to a File

Notes: Redirecting Output to a File

- Output from the SHOW PROCESS command can be displayed at your terminal or redirected to another output device. In this example, first we see the process information displayed on the screen, and then redirected to a file (PROCESS.TXT). We can then type the file to verify that the redirection was successful.

- The output redirection is reversed with the DEASSIGN command. Note that the DEASSIGN command does not aim SYS$OUTPUT back to the terminal; it simply removes the previous assignment .

Terminal I/O

Terminal I/O allows you to write command procedures that interact with the user. You can prompt the user to type information at the keyboard and then use that information in the command procedure. You can process information typed at the keyboard and send it to the user for verification.

Terminal input and output can be used to:

- Prompt the user for information
- Allow the use of an interactive utility, such as an editor
- Display messages and command output on the terminal screen
- Redirect terminal output to a file

A simple example of terminal I/O is the creation and use of a menu from which a user can select choices. You can create a menu by displaying formatted text on the screen. When the user makes a choice by typing the number or letter of one of the selections displayed, the command procedure will execute the task requested.

Controlling Terminal I/O

You control terminal I/O by redefining the same logical names that DCL uses to obtain and display information for your process. These logical names are used interactively to communicate with your process when you are typing at your terminal.

In similar fashion, logical names can be used to enable communication between the command procedure and the system.

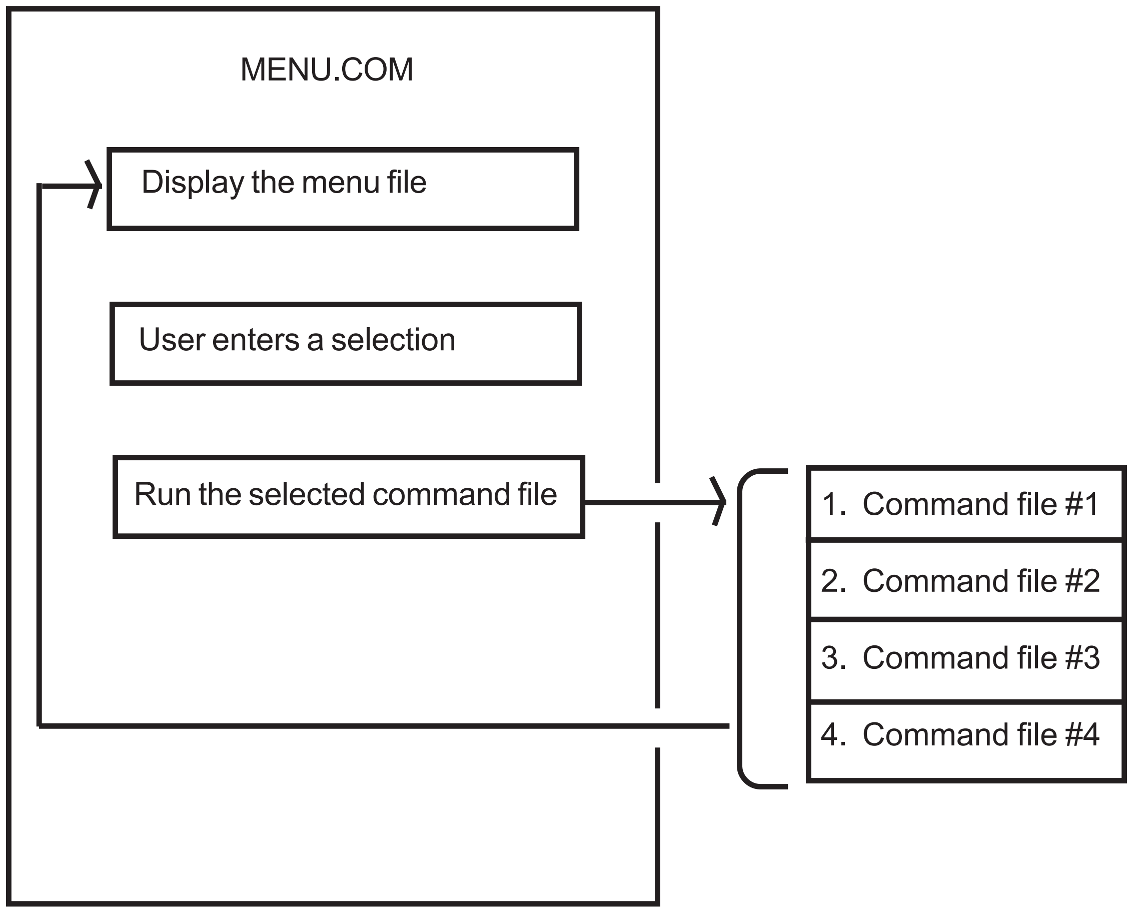

Figure 11-1 - How Menu Selection Can Be Used to Run Command Procedures

The values of logical names can be redefined to control terminal I/O in a command procedure. Files associated with terminal I/O do not have to be opened or closed; DCL opens and closes them automatically.

Logical Name	Description	Interactive	From a Command Procedure
SYS$COMMAND	Where the system expects commands to originate	Terminal	Terminal
SYS$INPUT	Where the system expects to receive input data	Terminal	Command procedure file
SYS$OUTPUT	Where the system expects to display the results of any command	Terminal	Terminal
SYS$ERROR	Where the system expects to write error messages	Terminal	Terminal

Table 11-2 - Logical Names Used with Terminal I/O

Displaying Information for the User at the Terminal

The WRITE or TYPE command will display information at your terminal. The information displayed can be:

- Symbol names
- Character strings in quotation marks
- Literal numeric values
- The results of lexical functions

The WRITE Command

The WRITE command will send a character string expression to an output device, such as SYS$OUTPUT. The syntax of the WRITE command to the terminal is:

```
WRITE  SYS$OUTPUT    expression
```

The expression specifies the data to be written as a single record. You can specify a list of expressions, separated by commas. The command interpreter concatenates the items into one record.

The expression can be one of the following (an example of each is included):

- A character string enclosed in quotation marks
  ```
  $  WRITE  SYS$OUTPUT  "Hello  World"
  ```
- A symbol name (the symbol's value is automatically substituted)
  ```
  $  WRITE  SYS$OUTPUT   X
  ```
- A lexical function
  ```
  $  WRITE  SYS$OUTPUT  F$TIME()
  ```
- A combination of items separated by commas
  ```
  $  WRITE  SYS$OUTPUT  "The  sum  is  ",X
  ```

Many SuRPAS programmers create a symbol to replace WRITE SYS$OUTPUT. It is not uncommon to find the following symbol assignment in a LOGIN.COM file or at the beginning of a command procedure:

```
$  SAY  :==  WRITE  SYS$OUTPUT
```

The example at the top of the next page illustrates the use of the WRITE command to display a user menu.

```
$ SAY :== WRITE SYS$OUTPUT
$ !
$ SAY " "
$ SAY " "
$ SAY " "
$ SAY "                    SELECT AN ITEM: "
$ SAY " "
$ SAY " "
$ SAY "          1. ENTER DATA AT THE TERMINAL"
$ SAY " "
$ SAY " "
$ SAY "          2. ENTER NAME OF DATA FILE "
$ SAY " "
```

Example 11-3 - Using SAY instead of WRITE SYS$OUTPUT to Display a User Menu

Notes: Using WRITE SYS$OUTPUT to Display a User Menu

- When the output expression consists of two quotation marks separated by a blank, a blank line is displayed.
- Character strings are displayed by enclosing them in quotation marks.

The output produced by the series of WRITE SYS$OUTPUT commands issued in the previous example can be seen below.

```



               SELECT AN ITEM:


      1.   ENTER DATA AT THE TERMINAL


      2.   ENTER NAME OF DATA FILE

```

Example 11-4 - Output From the WRITE SYS$OUTPUT Command

The TYPE Command

The TYPE SYS$INPUT command is often a difficult concept to grasp. The question most frequently asked is “Why TYPE instead of WRITE?”.

The explanation refers to the initial discussion of system logical names and how they are used to perform input and output. Initially, the system looks at SYS$INPUT for input. When interactive, it looks at the terminal; when a command procedure, it looks at the command procedure file.

So, if you want to display characters on the screen from a command procedure, you need to issue a command to do so (in this case TYPE) and then you need to give the command something to act on (in this case whatever it finds in SYS$INPUT - the command file). The system will continue to TYPE whatever it finds in SYS$INPUT (the command file) until it sees a $. The $ is an indicator to the system that the following characters begin another command.

Because the data in the TYPE SYS$INPUT command is treated as a literal:

- TYPE will not translate a symbol in the literal, and
- it will only successfully display the “$” character if it is **not** the first character in a line.

The format of the TYPE SYS$INPUT command is:

```
$ TYPE  SYS$INPUT
data
data
data
$
```

The TYPE SYS$INPUT command can be used to produce the same sample menu that was produced with the WRITE SYS$OUTPUT command.

```
$ TYPE SYS$INPUT



                         SELECT AN ITEM:


            1.   ENTER DATA AT THE TERMINAL


            2.   ENTER NAME OF DATA FILE

$
```

Example 11-5 - Using TYPE SYS$INPUT to Display a User Menu

Getting Information From the User

Getting information from the user is accomplished by using either the INQUIRE command or the READ command.

The INQUIRE command is generally used only for terminal I/O. The information from the INQUIRE command is converted to all uppercase characters and placed in either a local symbol or a global symbol (if you supply the qualifier /GLOBAL).

The READ command can be used for both terminal I/O and file I/O. The READ command allows you to supply a character string prompt when issuing the command. If you omit the prompt, the READ command prompts with the string `Data: .`

There are other differences between the INQUIRE command and the READ command. When doing file I/O, the READ command will trap the Ctrl/Z character combination to support the /END and /ERROR qualifiers; INQUIRE does not. While the READ command can be used in a captive account, the INQUIRE command cannot.

The INQUIRE Command

The INQUIRE command allows you to solicit information from the user. The syntax for the INQUIRE command is:

```
    $ INQUIRE  symbol-name   ["prompt-string"]
```

In the INQUIRE command, the optional ***prompt-string*** is displayed by the command prior to accepting input from the user. This allows you to customize the user prompt to fit the circumstances. The ***prompt-string*** will be followed by a colon (:) unless the /NOPUNCTUATION qualifier is included in the command. If the ***prompt-string*** is omitted, no prompt will be displayed (other than the colon).

All input from the user is converted to uppercase characters and stored in a local symbol (***symbol-name***) unless the INQUIRE command is issued with the /GLOBAL qualifier. Multiple spaces and tabs are replaced by single spaces as a part of the conversion.

The INQUIRE command is used extensively in SuRPAS to determine the existence of a file.

The READ Command

The READ command has the following syntax:

```
$ READ[/PROMPT=string] SYS$COMMAND symbol-name
```

The READ command does **not** convert the user's response to uppercase and multiple spaces are **not** removed. The user's response is stored in the ***symbol-name*** variable.

In the following example the TYPE SYS$INPUT command is used to demonstrate both the INQUIRE and the READ commands.

```
$!                  I O . C O M
$!        Demonstration of simple terminal I/O
$ TYPE SYS$INPUT
Using INQUIRE

$ INQUIRE YOU1 "Please type your name"
$ WRITE SYS$OUTPUT YOU1," is using this terminal."
$ WRITE SYS$OUTPUT " "  ! Blank line
$!
$!
$ WRITE SYS$OUTPUT "Using READ"
$ WRITE SYS$OUTPUT " "
$ READ/PROMPT="Please type your name" SYS$COMMAND YOU2
$ WRITE SYS$OUTPUT YOU2," is using this terminal."
$ WRITE SYS$OUTPUT " "
$ EXIT
$!
$! Sample run of IO.COM
$!
$ @IO
Using INQUIRE

Please type your name:  Suzy Homaker
SUZY HOMAKER is using this terminal.

Using READ

Please type your name Suzy Homaker
Suzy Homaker is using this terminal.

$
```

Example 11-6 - Terminal I/O in Command Procedures

File I/O

The OpenVMS operating system supports file I/O through the use of Record Management Services (RMS) routines. There are four basic file I/O commands supported in the form of DCL commands.

- The OPEN command tells OpenVMS to prepare the file for I/O. This involves setting up data structures and fetching file characteristics.
- The READ and WRITE commands tell OpenVMS to perform the I/O.
- The CLOSE command tells OpenVMS to remove the data structures from memory.

The basic steps in reading and writing files from a command procedure are:

1. Use the OPEN command to open the file.
2. Use the READ or WRITE commands to read or write records.
   - Read a record.
   - Perform some task on the record.
   - Write the record.
   - Read the next record, and continue with the previously mentioned steps until you have reached the end of the file.
3. Use the CLOSE command to close the file. Unless you explicitly close the file, it will remain open until you log out.

Opening A File

The OPEN Command

Using the DCL OPEN command, a file can be opened for reading (default), writing, or for both reading and writing. The OPEN command assigns a logical name to the file and defines whether or not the file is shareable by more than one user. The following is the syntax for the OPEN command:

```
$ OPEN[/qualifiers] logical-name[:] file-
specification
```

The ***file-specification*** should always be entered in its complete form so that it will not be necessary for the user to be in the directory when running the command procedure. Wildcards are not permitted in the file-specification, and the file type defaults to .DAT. The following files do not have to be explicitly opened before they can be read or written: SYS$INPUT, SYS$OUTPUT, SYS$COMMAND, and SYS$ERROR. The ***file-specification*** can be a symbol name if it is surrounded in a pair of single quotes (e.g. ‘INFILE’).

The ***logical-name*** allows you to easily differentiate between multiple open files. Do not use the same logical-name when opening different files. If you use the /SHARE qualifier the first time a file is opened, you can enter more than one OPEN command for the same file, and assign it different logical names.

- If you use /SHARE=READ, other users are allowed read access to the file.
- If you use /SHARE=WRITE, other users are allowed read and write access to the file.

Use ***qualifiers*** to specify the type of access you desire. The /READ and /WRITE qualifiers do not necessarily restrict your access to a file. They determine the error handling if the file does not exist.

- The /READ (default) tells the command to open an existing file.
- The /WRITE tells the command to open a *new* file or a *new version* of an existing file.
- The /APPEND opens an existing file and adds records to the end of it. The /APPEND qualifier cannot be used with the/WRITE qualifier.
- The /ERROR=label qualifier transfers control to a label location if an error occurs.
- The /SHARE qualifier allows multiple users to open a single file.

Reading From a File

The READ Command

The READ command reads a single record from a specified input file and assigns its contents to a specified symbol name. The syntax for the READ command is:

```
$ READ[qualifier(s)] logical-name[:]
symbol-name
```

The input file for the READ command can be of type sequential, relative, or indexed sequential. After each record is read from the specified file (as indicated by the ***logical-name***), the record pointer is positioned at the next record in the file. If you are reading an indexed file, you can use the /INDEX and /KEY qualifiers to read records randomly. The maximum record size for a single READ is 2048 bytes.

The READ command accepts data exactly as it is entered. It does not convert characters to uppercase, remove extra spaces or tabs, or remove question marks. It also does not perform symbol substitution. The data record read is assigned to the specified ***symbol-name***.

Other ***qualifiers*** available to the READ command include:

- The /END_OF_FILE=*label* qualifier, used within the context of a loop to determine the location (as specified by the supplied ***label***) to which control will be transferred after the last record in the file is read.
- The /ERROR=*label* qualifier, which determines the location (as specified by the supplied ***label***) to which control will be transferred if an error occurs.

```
$ OPEN/READ INFILE SAMPLE.DAT
$ READ_LOOP:
$        READ/END_OF_FILE=ENDIT INFILE RECORD
.
.
.
$        GOTO READ_LOOP
$ ENDIT:
$        CLOSE INFILE
$        EXIT
```

Example 11-7 - The READ Command

Writing to A File

The WRITE Command

The WRITE command writes data as one record to an open file specified by a logical name. The syntax for the WRITE command is:

```
$ WRITE[/qualifier(s)] logical-name[:]
expression
```

The ***expression*** represents data as a single record when it is output to the file. A list of expressions separated by commas will be concatenated into one record. The maximum size for any record that can be written is 1024 bytes. If you specify the /SYMBOL qualifier, the maximum record size is 2048 bytes. Use the /SYMBOL when the record contains 1000 bytes or more. Each expression specified must be a symbol.

Other qualifiers available for the WRITE command include the /ERROR and /UPDATE qualifiers.

- The /ERROR qualifier defines a label location where control is transferred when an error occurs. The /ERROR qualifier overrides any existing ON condition action previously specified.
- The /UPDATE replaces the last record you have read. It is only used after a READ command. When using sequential files, you must replace the record with another record of the same size.

```
$ OPEN/WRITE OUT DATA.NEW
$ WRITE_LOOP:
$         INQUIRE DATA
$         IF DATA .EQS. “END” THEN GOTO QUIT
$         WRITE/SYMBOL OUT DATA
$         GOTO WRITE_LOOP
$!
$ QUIT:
$         CLOSE OUT
$         EXIT
$
```

Example 11-8 - The WRITE Command

Closing a File

The CLOSE Command

The DCL CLOSE command closes an open file and deassigns the associated logical name. If an open file is not explicitly closed with this command, it remains open until one of the following occurs:

- the termination of the process that opened it ,or
- the user logs off.

The syntax for the CLOSE command is:

```
$ CLOSE[/qualifier(s)]  logical-name[:]
```

There are a number of qualifiers available for use with the CLOSE command. They include:

- The /ERROR=*label* qualifier, which determines the location (as specified by the supplied ***label***) to which control will be transferred if an error occurs.
- The /LOG qualifier, which causes the display of a warning message that indicates that the file attempting to be closed was not opened by DCL. This qualifier is ignored if you specify the /ERROR qualifier. (The /NOLOG qualifier is also available for this command.)

An additional note about the CLOSE command

The fact that exiting from a command procedure does not close an open file has caused much confusion while debugging command procedures.

- A nonfile-related error terminates the procedure, so the programmer edits the procedure.
- Although the edit fixes the problem, the procedure does not run properly because the file is partially processed with the record pointer somewhere in the middle of the file.

More Uses for System-Created Logical Names

There are additional uses for system-created logical names and their ability to be redefined.

We can redirect SYS$INPUT to SYS$COMMAND when we want to invoke the editor or run a program from within a command procedure. To accomplish this, we must have an interactive session that can communicate with the editor or program. We can make this happen by redirecting our input to the input device used by the editor or program.

The OpenVMS operating system supports user level (or user-mode) logical names. Until now we have focused on logical names that were at the process level. The user-mode logical name only remains in existence for the duration of a single image execution. It is deleted from the process logical name table when the image terminates.

In earlier lessons we discussed using SYS$INPUT and SYS$OUTPUT to send and receive information in command procedures. In this section, we will talk about redefining these two system-created logical names to redirect input and output.

Redefining SYS$INPUT

You can redefine SYS$INPUT to allow a command procedure to read from the terminal or another file. Normally, the system will only read input from the command file itself; by redefining SYS$INPUT, you can force the system to read instructions from another file or the terminal.

For example, to edit a file from a command procedure, include the following lines in the command procedure:

```
$  DEFINE/USER_MODE  SYS$INPUT  SYS$COMMAND
$  EDIT  MYFILE.DAT
```

Editors obtain input from SYS$INPUT, which is the command procedure, in this case. SYS$COMMAND refers to the terminal, the initial input device when you logged in. This DEFINE command directs your input to the editor.

In the example above, the /USER_MODE qualifier is used to tell the command procedure that the SYS$INPUT redirection is not a long-term activity; it is in effect only for the duration of the next executing image (the editor). This allows you to perform edits interactively. When the editor image exits, SYS$INPUT resumes its default value within the command procedure.

You can accomplish the same result when communicating to an interactive program from a command procedure. Issue the identical DEFINE command as in the above example, and then RUN your interactive program. When the program image exits, control returns to the command procedure. Those command procedure statements might appear as follows:

```
$  DEFINE/USER_MODE  SYS$INPUT  SYS$COMMAND
$  RUN  MY_DAILY_BACKUP
```

Redefining SYS$OUTPUT

When you issue a command that results in output (e.g. the DIRECTORY command), the results are displayed on your screen. (When you issue this type of command in a command procedure, the default for SYS$OUTPUT remains on the screen.) It may be necessary to capture the output and do something with it, such as turning it into a report. To do this, you have to redefine SYS$OUTPUT so that the results go to a file instead of the screen. As with the redefining of SYS$INPUT in the previous section, use of the /USER_MODE qualifier allows you make this change for the duration of the next command only. Once the command is completed, SYS$OUTPUT resumes its default value.

In the following two command lines, the display produced by the SHOW DEVICES command is directed to MY_DEVICE_FILE.DAT rather than the terminal screen.

```
$  DEFINE/USER_MODE  SYS$OUTPUT  MY_DEVICE_FILE.DAT
$  SHOW  DEVICES
```

Using File I/O To Display A Text File - A Student Lab

In Chapter Five you created two WISHLIST.TXT files. One file contains a list of your top ten decadent items and the second, a list of your needed items.

Using the File I/O commands discussed in this chapter (OPEN, CLOSE, READ, WRITE, TYPE, etc.), create a command file that will:

1. **Open each of your WISHLIST.TXT files.**

2. Create a new file for writing.

3. **Read in a line of text from each file.**

4. Write each line of text to the display screen.

5. **Then write the two lines to the newly created output file**.

6. Loop back and Read in the next line of text from each file, until you reach the end-of-file.

7. **Place a counter in the READ-WRITE loop to count the number of lines read from each file and written out.**

8. Close the three WISHLIST.TXT files.

9. **Display the final count of total number of lines and notification that the READ-WRITE activity has completed successfully.**

10. Be sure to test for OPEN, READ, and WRITE errors. If the activity aborts prematurely, due to an error, close the files and Display an error notification and the number of lines successfully written prior to the error.

Summary

Concepts

- Use Terminal I/O to gather and use information typed at the terminal in a command procedure.
  - Use WRITE SYS$OUTPUT to display information to the terminal screen, one line at a time.
  - Use TYPE SYS$INPUT to display information from several lines in the command file on the terminal screen.
  - Use the INQUIRE and READ commands to obtain information from the user in command procedures.

- Use File I/O to gather and use information stored in a file.
  - Use the OPEN and CLOSE commands to open and close for reading and writing.
  - Use the READ and WRITE commands to then get information from or send information to a file from a command procedure.

- Redefine SYS$INPUT and SYS$OUTPUT to redirect where information or instructions are displayed or read.
  - Use the /USER_MODE qualifier to limit the duration of the redefinition.