EDUCATIONAL SERVICES

Utilizing VMS Features from VAX COBOL

Language-Specific Workbook

digital

EY-2322E-WB-0001

Utilizing VMS Features from VAX COBOL

Language-Specific Workbook

Prepared by Educational Services
of
Digital Equipment Corporation

First Edition, November 1984

Printed in U.S.A.

Calling Procedures

Tables

Examples

Synchronizing Processes

Figures

Examples

Accessing Devices

Examples

Communicating with Other Processes

Examples

Creating and Managing Other Processes

Examples

Gathering and Displaying Data at Terminals

Examples

Building Human Interfaces

Examples

Creating, Accessing, and Ordering Files

Tables

Examples

Handling Exceptions and Exits

Examples

Sharing Code and Data

Examples

Measuring and Improving Performance

Examples

Calling Procedures

File Location

On-line files are provided to help you master this module. The files are located in the directory:

```
DISK$COURSE:[COURSE.V4PROG.COB.CALL]
```

For your convenience, your system manager may have created the following logical name equivalence

```
DISK$COURSE:[COURSE.V4PROG.COB.CALL] = V4PROG$COB$CALL
```

Three types of files are provided:

1. The Module Programming Examples, which are also listed in this language workbook.
2. Files that you modify to complete the Laboratory Exercises.
3. Solutions to the Laboratory Exercises.

If you wish to modify and/or compile, link, and run any of the programs, first copy the file into your directory.

Calling External Procedures From COBOL

You can call an external procedure as a subroutine or as a function as shown in Table 1.

Table 1 Format of Subroutine and Function Codes

Calling Method	Example
SUBROUTINE	CALL 'SUB1' USING ARG1, ARG2.
FUNCTION	CALL 'SUB1' USING ARG1, ARG2 GIVING RESULT.

System supplied procedures are documented in the *VAX/VMS System Routines Reference Volume*, which contains the following manuals:

1. *Introduction to System Routines*
2. *VAX/VMS System Services Reference Manual*
3. *VAX/VMS Run-Time Library Routines Reference Manual*
4. *VAX/VMS Record Management Services Reference Manual*
5. *VAX/VMS Utility Routines Reference Manual*

Argument Passing Mechanisms

Procedures written in COBOL can only accept arguments passed by reference. COBOL has two mechanisms that perform a by reference pass: **BY REFERENCE** and **BY CONTENT**.

The **BY REFERENCE** mechanism passes the adddress of the argument to the called program. The called program references the same storage area for the data item as the calling program.

The **BY CONTENT** mechanism copies the argument to a temporary location, and passes the address of the temporary location to the called program. This mechanism ensures that the called program cannot change the original contents of the argument.

A COBOL program can call a procedure written in other languages even if the procedure accepts its arguments by methods conflicting with the COBOL default. The default can be overridden using the mechanisms listed in Table 2.

Table 2 Argument Passing Mechanisms

Passing Method	Mechanisms	Description*
Immediate Value	BY VALUE	"the value" or omission of the word "address"
Reference	BY REFERENCE or BY CONTENT	"address of"
Descriptor	BY DESCRIPTOR	"address of a descriptor"

* These keywords are used in descriptions in the *VAX/VMS System Routines Reference Volume*.

Examples 1 through 3 show how a COBOL program passes arguments to an external procedure by each of the three methods.

```
CALL 'SUB1' USING BY VALUE ABC
```

Example 1 Passing Arguments by Value

```
CALL 'SUB2' USING BY REFERENCE ABC
        OR
CALL'SUB2' USING ABC
```

Example 2 Passing Arguments by Reference

```
CALL 'SUB2' USING BY DESCRIPTOR ABC
```

Example 3 Passing Arguments by Descriptor

LEARNING ACTIVITY

Refer to the *VAX COBOL User's Guide* for more information about calling procedures from a COBOL program.

Calling Procedures Written in Other Languages

In addition to obeying the VAX/VMS Procedure Calling Standard, calling programs and their called procedures must agree on:

- The data type of each argument (for example, whether the data is floating-point or integer).
- The method by which each argument is passed. Different languages have different default passing mechanisms.

Table 3 compares the default passing mechanisms for different native-mode languages.

Table 3 Default Passing and Receiving Mechanisms

Language	Numeric	Character	Arrays
COBOL	Reference	Reference	N/A
BASIC	Reference	Descriptor	Descriptor
FORTRAN	Reference	Descriptor	Reference
MACRO	No Default	No Default	No Default

Example 4 shows a COBOL main program and functions written in other native-mode languages.

❶ The COBOL main program SHOWSUM calls a function GETSUM to calculate the sum of two integers. Numeric data is passed by reference by all of the languages.

❷ Notice that all of the functions are called GETSUM. The COBOL main program will run with any function.

❸ The COBOL main program SHOWSUM, and the BASIC function GETSUM, are compiled and linked. The execution of the program follows.

```
 1       *                                                SHOWSUM.COB
 2       IDENTIFICATION DIVISION.
 3       *
 4       PROGRAM-ID. SHOWSUM.
 5       *
 6       *   This program can call a function written in another
 7       *   native mode language.  The program requests 2 numbers,
 8       *   calls a function to calculate the sum, and prints the
 9       *   result.  The program stops if ^Z is entered for input.
10       *
11       DATA DIVISION.
12       WORKING-STORAGE SECTION.
13
14       01  A        PIC 9(9)        COMP.
15       01  B        PIC 9(9)        COMP.
16       01  RESULT   PIC 9(9)        COMP.
17       01  IN-A     PIC 9(3).
18       01  IN-B     PIC 9(3).
19       01  DIS-RES  PIC Z(4).
20
21       PROCEDURE DIVISION.
22       BEGIN.
23           DISPLAY 'Please enter an integer, type ^Z to stop '
24                   WITH NO ADVANCING.
25           ACCEPT IN-A AT END STOP RUN END-ACCEPT.
26           DISPLAY 'Please enter an integer, type ^Z to stop '
27                   WITH NO ADVANCING.
28           ACCEPT IN-B AT END STOP RUN END-ACCEPT.
29           MOVE IN-A TO A.
30           MOVE IN-B TO B.
31   (1)     CALL 'GETSUM' USING A, B GIVING RESULT.
32           MOVE RESULT TO DIS-RES.
33           DISPLAY 'The sum is ', DIS-RES.
34           GO TO BEGIN.


 1       10!                                              BASSUM.BAS
 2         !
 3         ! This BASIC function calculates the sum of two integers.
 4         !
 5   (2)     FUNCTION INTEGER GETSUM (INTEGER A,B)
 6           GETSUM = A + B
 7           FUNCTIONEND


 1       C**                                              FORSUM.FOR
 2       C
 3       C   This FORTRAN function calculates the sum of two
 4       C   integers.
 5       C
 6   (2)     INTEGER FUNCTION GETSUM(A,B)
 7           IMPLICIT INTEGER (A - Z)
 8           GETSUM = A + B
 9           RETURN
10           END
11
```

Example 4 A COBOL Main Program and Functions Written in Other Native Mode Languages (Sheet 1 of 2)

```
1       ;                                               MACSUM.MAR
2       ;
3       ; This MACRO function calculates the sum of two numbers.
4       ;
5    ❷      .ENTRY  GETSUM,^M<>
6           CLRQ    R0                      ;CLEAR R0 and R1
7           ADDL3   @4(AP),@8(AP),R0        ;OBTAIN SUM
8           RET
9           .END
```

```
❸ $ COBOL SHOWSUM
  $ BASIC BASSUM
  $ LINK SHOWSUM, BASSUM
  $ RUN SHOWSUM
  Please enter an integer, type ^Z to stop 057
  Please enter an integer, type ^Z to stop 089
  The sum is  146
  Please enter an integer, type ^Z to stop 172
  Please enter an integer, type ^Z to stop 024
  The sum is  196
  Please enter an integer, type ^Z to stop 009
  Please enter an integer, type ^Z to stop 018
  The sum is   27
  Please enter an integer, type ^Z to stop ^Z
  $
```

Example 4 A COBOL Main Program and Functions Written in Other Native Mode Languages (Sheet 2 of 2)

Testing Status in COBOL

The VAX/VMS Procedure Calling Standard requires that a status code be returned from an external procedure to indicate success or failure. The status code is returned to the calling program as a function result.

To use the status code from a COBOL program you must:

- Call the external procedure as a function.
- Declare the variable to receive the status code. It must be COMP usage, in the range of S9(5) to S9(9).

You can check for success or failure of an external procedure with a success/failure test of the status value as shown in Example 5. You can also check for specific codes as shown in Example 6.

```
          .
          .
          .
DATA DIVISION.
WORKING-STORAGE SECTION.
01          STAT        PIC S9(9)        COMP.
PROCEDURE DIVISION.
          .
          .
      CALL 'SUB1' USING ... GIVING STAT.
      IF STAT IS FAILURE handle error
          .
          .
          .
```

Example 5 Checking for Procedure Success or Failure

```
          .
          .
          .
DATA DIVISION.
WORKING-STORAGE SECTION.
01 STAT      PIC S9(9) COMP.
01 status_code PIC S9(9) COMP  VALUE IS EXTERNAL status_code.
          .
          .
PROCEDURE DIVISION.
BEGIN.

      CALL 'SUB1' USING ... GIVING STAT.
      IF STAT IS EQUAL TO status_code THEN handle error
          .
          .
```

Example 6 Checking for Specific Status Returns

Status codes for system services, Run-Time Library procedures, and record management services follow the naming convention facility__code$__status__code. Specific status codes are listed with each procedure in the appropriate reference manuals.

References to status code symbols can be resolved two ways. You can incorporate the status code symbol definitions into a file with the EXTERNAL clause. Status code symbols defined this way are resolved when the linker searches the system object library SYS$LIBRARY:STARLET.OLB.

Example 7 uses the EXTERNAL clause to resolve the reference to a return status code symbol in a call to the $CLREF system service. A check is made for the SS$__ILLEFC return status code. The $CLREF system service accepts the number of an event flag and clears it.

```
      .
      .
      .
DATA DIVISION.
WORKING-STORAGE SECTION.
01        EFN                 PIC 9(9)          COMP VALUE 1.
01        STAT                PIC S9(9)         COMP.
01        SS$_ILLEFC          PIC S9(9)         COMP VALUE EXTERNAL SS$_ILLEFC.
      .
      .
      .
PROCEDURE DIVISION.
 BEGIN.
      CALL 'SYS$CLREF' USING BY VALUE EFN
                       GIVING STAT.
      IF STAT IS EQUAL TO SS$_ILLEFC THEN handle error
      .
      .
      .
```

Example 7 Using the EXTERNAL Clause in a Program

You can also assign a value to each symbol with the VALUE clause. Since this method requires coding fixed numeric codes into the program, it is recommended only if the EXTERNAL clause will not resolve the reference. If the codes change in future releases of VMS, the programs containing fixed numeric codes will require modification.

Status code variables are defined by the macros $LIBDEF, $RMSDEF and $SSDEF, in the system library, SYS$LIBRARY:STARLET. To determine the symbol values:

1. Create a file SSDEF.MAR:

   ```
   $SSDEF GLOBAL
   .END
   ```

2. Issue the command $ MACRO/LIST SSDEF.
3. Issue the command $TYPE SSDEF. The symbol table contains the hexidecimal values of all SS$ status codes. The value must be converted to decimal before using it in a COBOL program.
4. Use the VALUE caluse to define the code. For example, to define the code for SS$__ILLEFC:

   ```
   01 SS$_ILLEFC    PIC S9(9) COMP    VALUE 252
   ```

Calling System-Supplied Procedures

System supplied procedures are documented in the *VAX/VMS System Routines Reference Volume*, described previously in this module.

Calling Run-Time Library Procedures

The Run-Time Library is part of the system library that is automatically searched when user programs are linked by the $LINK command. To call a Run-Time Library procedure, the order of the arguments must follow the order shown in the *VAX/VMS Run-Time Library Routines Reference Manual*.

Some procedure arguments are described as optional. If you do not wish to provide specific data for an optional argument, you may provide a "dummy" argument. The "dummy" argument should pass a zero BY VALUE, regardless of the stated mechanism for the optional argument. An omitted argument is detected when a value of zero is given for a specific entry on the list.

If an optional argument is passed BY REFERENCE, it is interpreted as the address of the optional argument, which is nonzero. When an optional argument is passed BY REFERENCE, the list entry is considered to be a valid argument (not an omitted argument).

Example 8 shows a call to the Run-Time Library procedure LIB$PUT_OUTPUT to write an ASCII string to SYS$OUTPUT. The argument must be passed BY DESCRIPTOR.

```
 1  *
 2  *
 3  IDENTIFICATION DIVISION.
 4  *
 5  PROGRAM-ID.  LIB_PUT_OUTPUT.
 6  *
 7  *       This procedure will invoke the VAX/VMS
 8  *       Run-Time Procedure LIB$PUT_OUTPUT to
 9  *       output a character string to SYS$OUTPUT.
10  *
11  DATA DIVISION.
12  WORKING-STORAGE SECTION.
13  PROCEDURE DIVISION
14  BEGIN.
15          CALL 'LIB$PUT_OUTPUT' USING
16                  BY DESCRIPTOR 'HELLO THERE'.
```

Example 8 Calling a Run-Time Library Procedure

Calling RMS Procedures

To call an RMS procedure, the order of the arguments in the call must correspond with the order shown in the *VAX RMS Reference Manual*. Reserve a place on the call statement for every argument. As when calling Run-Time Library procedures, pass optional arguments as a zero BY VALUE, regardless of the stated mechanism for the argument.

The procedure name format is SYS$procedure_name when calling an RMS routine from COBOL. Example 9 calls the RMS procedure $SETDDIR. This RMS procedure sets the default directory for a process.

❶ The directory name is assigned to an alpha-numeric variable.

❷ The call to $SETDDIR contains the DIRECTORY name which is passed BY DESCRIPTOR. The optional arguments pass a zero BY VALUE to reserve space in the argument list.

❸ Program examples in this course check the status return after each call to an external procedure. If the return status code is unsuccessful, the call to LIB$STOP will stop the program execution and print the reason for failure.

❹ A $SHOW DEFAULT command shows that the default directory is set to WORK2:[COURSE.V4PROG.COB.CALL], which contains the file SETDDIR.COB.

❺ The program is executed.

❻ Another $SHOW DEFAULT command shows that the default directory has changed. The directory is now set to WORK2:[COURSE.PROG.COB].

```
                                                        SETDDIR.COB
      IDENTIFICATION DIVISION.
      *
      PROGRAM-ID. SETDDIR.
      *
      *   This program calls the RMS procedure $SETDDIR to change
      *   the default directory for the process.
      *
      DATA DIVISION.
      WORKING-STORAGE SECTION.

  ❶   01  DIRECTORY PIC  X(17)              VALUE
                                            '[COURSE.PROG.COB]'.
      01  STAT      PIC S9(9)     COMP.

      PROCEDURE DIVISION.
      BEGIN.
  ❷       CALL 'SYS$SETDDIR' USING BY DESCRIPTOR DIRECTORY
                                    BY VALUE 0
                                    BY VALUE 0
                               GIVING STAT.
  ❸       IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
          STOP RUN.

$ COBOL SETDDIR
$ LINK SETDDIR
$
❹ $ SHOW DEFAULT
  WORK2:[COURSE.V4PROG.COB.CALL]
$
❺ $ RUN SETDDIR
❻ $ SHOW DEFAULT
  WORK2:[COURSE.PROG.COB]
$
```

Example 9 Calling the RMS Procedure $SETDDIR

Calling the CLI

Example 10 transfers control to the command language interpreter using the LIB$DO__COMMAND Run-Time Library procedure. The LIB$DO__COMMAND procedure executes a DCL command or command procedure after terminating the image. Upon completion of the command, control is **permanently** transferred to the command language interpreter.

❶ The LIB$DO__COMMAND can execute a command procedure. The VALUE clause contains the DCL command to execute a command procedure called LISTIT.

❷ The argument must be passed by DESCRIPTOR.

❸ This command procedure executes when LIB$DO__COMMAND is called.

❹ After compiling and linking DOCOMMAND.COB, the program is executed. The command procedure LISTIT.COM is executed and the two messages are typed.

```
 1        *                                              DOCOMMAND.COB
 2        IDENTIFICATION DIVISION.
 3        *
 4        PROGRAM-ID. LIB-DO-COMMAND.
 5        *
 6        *   This program calls the command language interpreter with
 7        *   the LIB$DO_COMMAND.  A command procedure is executed.
 8        *
 9.       DATA DIVISION.
10        WORKING-STORAGE SECTION.
11
12   ❶    01  COM             PIC X(7)        VALUE '@LISTIT'.
13
14        PROCEDURE DIVISION.
15        BEGIN.
16   ❷        CALL 'LIB$DO_COMMAND' USING BY DESCRIPTOR COM.
17        *
18        *   If the call to LIB$DO_COMMAND is successful,
19        *   control is not returned to the main program.
20            DISPLAY 'Error in the call to LIB$DO_COMMAND'.
21            STOP RUN.


 1   ❸    $!                                             LISTIT.COM
 2        $WRITE SYS$OUTPUT "Control is passed to a command procedure"
 3        $WRITE SYS$OUTPUT "This is a test of LIB$DO_COMMAND"
 4        $EXIT

$ COBOL DOCOMMAND
$ LINK DOCOMMAND
❹ $ RUN DOCOMMAND
Control is passed to a command procedure
This is a test of LIB$DO_COMMAND
$
```

Example 10 Calling the Command Language Interpreter Using the Run-Time Library Procedure LIB$DO__COMMAND

Calling System Services

Although system services were written for use by the operating system, they are also useful in applications programs.

The order of the arguments in the call must correspond with the order shown in the *VAX/VMS System Services Reference Manual*. You must reserve a place in the call for every argument. As when calling Run-Time Library procedures and RMS procedures, pass optional arguments as a zero BY VALUE, regardless of the stated mechanism for the argument.

The procedure name format is SYS$procedure_name when calling a system service from COBOL. Example 11 shows a call to the system service $GETMSG, which returns message text associated with a given message identification code. It shows the three mechanisms for passing arguments to an external procedure.

❶ Definitions for variables passed to the $GETMSG system service are located in the DATA DIVISION.

❷ The arguments are passed by different mechanisms:

a. The first argument (MSGID) is described as "a longword value containing the message identification," so the argument must be passed by value. The passing mechanism specifier BY VALUE is required since COBOL's default passing mechanism for numeric values is by reference.

b. The second argument (MSGLEN) is described as "Length of the message string returned by $GETMSG." The MSGLEN argument is the address of a word into which $GETMSG writes this length. The argument must be passed by reference, but the passing mechanism specifier BY REFERENCE is not required because numeric values are (by default) passed by reference.

c. The third argument (BUFADR) is described as "the address of a character string descriptor pointing to the buffer into which $GETMSG writes the message string." The argument must be passed by descriptor, but BY DESC is not required because character data is (by default) passed by descriptor.

d. The fourth argument (FLAGS) is described as "a longword bit vector." Since the word "address" is not used, the argument is passed BY VALUE.

e. The fifth argument (OUTADR) is described as "address of a four byte array," hence the argument is passed BY REFERENCE.

The GIVING clause on the CALL statement specifies where the status code will be returned. Then the program may be checked for specific status codes.

❸ A test is made for two specific error conditions, buffer overflow and message not found. The data names SS$_BUFFEROVF and SS$_MSGNOTFND are defined in the data division by the EXTERNAL clause.

```
                                                        GETMSG.COB
      IDENTIFICATION DIVISION.
      *
      PROGRAM-ID. GETMSG.
      *
      *   This program is an example using the 3 mechanisms to
      *   pass arguments to a system services. It also shows
      *   how to test for specific status codes from a system
      *   service call.
      *
      DATA DIVISION.
      WORKING-STORAGE SECTION.

  (1) 01  MESSAGE-ID          PIC 9(9)    COMP.
      01  MESSAGE-LENGTH      PIC 9(9)    COMP.
      01  MESSAGE-TEXT        PIC X(132).
      01  MASK                PIC 9(9)    COMP VALUE 15.
      01  OUT-ARRAY           PIC X(4).
      01  MESSAGE-ID-DIS      PIC X(4).
      01  MSG-1               PIC X(15)   VALUE 'BUFFER OVERFLOW'.
      01  MSG-2               PIC X(17)   VALUE 'MESSAGE NOT FOUND'.
      01  STAT                PIC S9(9)   COMP.
      01  SS$_BUFFEROVF       PIC S9(9)   COMP VALUE EXTERNAL
                                           SS$_BUFFEROVF.
      01  SS$_MSGNOTFND       PIC S9(9)   COMP VALUE EXTERNAL
                                           SS$_MSGNOTFND.
      PROCEDURE DIVISION.
      BEGIN.
          DISPLAY 'Enter a message number ' WITH NO ADVANCING.
          ACCEPT MESSAGE-ID-DIS AT END STOP RUN END-ACCEPT.
          MOVE MESSAGE-ID-DIS TO MESSAGE-ID.
          INITIALIZE MESSAGE-TEXT.
  (2)     CALL 'SYS$GETMSG' USING BY VALUE MESSAGE-ID
                                  BY REFERENCE MESSAGE-LENGTH
                                  BY DESCRIPTOR MESSAGE-TEXT
                                  BY VALUE MASK
                                  BY REFERENCE OUT-ARRAY
                           GIVING STAT.
          EVALUATE STAT
  (3)         WHEN SS$_BUFFEROVF DISPLAY MSG-1
              WHEN SS$_MSGNOTFND DISPLAY MSG-2
              WHEN OTHER DISPLAY MESSAGE-TEXT(1:MESSAGE-LENGTH)
          END-EVALUATE.
          GO TO BEGIN.
```

```
$ COBOL GETMSG
$ LINK GETMSG
$ RUN GETMSG
Enter a message number 0615
%SYSTEM-?-NOTAPEOP, no tape operator
Enter a message number 1617
%SYSTEM-S-CONTROLC, operation completed under CTRL/C
Enter a message number 1561
%SYSTEM-S-CREATED, file/section did not exist - was created
Enter a message number 1492
MESSAGE NOT FOUND
Enter a message number ^Z
$
```

Example 11 Calling the $GETMSG System Service

LEARNING ACTIVITY

Refer to the *VAX COBOL User's Guide* for more information about calling system services.

Avoiding Common Errors

When you debug calls to external procedures, check for the following common mistakes.

- Are the arguments passed by the correct method? The four mechanisms are by value, by reference, by descriptor, and by content. In a call statement, a specified argument passing mechanism applies to every argument following it, until a new mechanism appears.
- Does the argument list contain the correct number of arguments? Omitted optional arguments must have a place reserved for them on the argument list. Use BY VALUE 0 to pass all omitted arguments.
- Is the external procedure name in quotes on the CALL statement?
- Has the data name in the GIVING clause for the status code been declared in the DATA DIVISION as a COMP usage in the range 9(5) to 9(9)?
- Are status code symbol references resolved with an EXTERNAL clause in the DATA DIVISION?

Lab Exercises

1.

a. The following COBOL program calls a VAX BASIC subprogram to add two longword integers. By what passing mechanism does the subprogram expect to receive the passed parameters?_____

b. By what mechanism is the main program passing the parameters?_____

c. Copy these programs, then compile, link and run. Explain why the wrong result is obtained.

d. What should be the passing mechanism used by the main program?_____. Modify the main program to obtain the correct result.

Provided File

```
*                                                                LAB1.COB
IDENTIFICATION DIVISION.
*
PROGRAM-ID.    LAB1.
*
*      This procedure will invoke the BASIC subprogram
*      ADDIT to add two integers.
*
DATA DIVISION.
WORKING-STORAGE SECTION


  01 I      PIC  9(5)   COMP   VALUE  1.
  01 J      PIC  95     COMP   VALUE  2.

PROCEDURE DIVISION.
 BEGIN.
*
*      Invoke the BASIC subprogram ADDIT
       CALL 'ADDIT' USING BY DESCRIPTOR I J.
       STOP RUN.
```

```
!                                                                ADDIT.BAS
10     SUB ADDIT (I%, J%)
20     K% = I% + J%
30     PRINT "The sum of the two arguments is ", K%
40     SUBEND
```

2. Write a program that declares a character string variable containing the value "Hello there!" (quotes are not included). Use the VAX/VMS Run-Time Library procedure LIB$PUT_OUTPUT to output this string to SYS$OUTPUT (the controlling terminal). Do this in two different ways:

   a. Call LIB$PUT_OUTPUT as a subroutine. This method does not allow status checking for success or failure of the call.

   b. Call LIB$PUT_OUTPUT as a function, returning status information in a longword integer variable called STAT. Use the following line of code to test the value returned in STAT:

      IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT

      Try omitting all arguments in the call to LIB$PUT_OUTPUT.

3. Write a program that:
   a. Calls the Clear Event Flag service to clear event flag 2.
   b. Checks the status value for success.
   c. Calls LIB$STOP if the call fails, or prints the following message if the call succeeds:

      EFN 2 has been cleared.

      Repeat the above steps, using event flag 75 and again using event flag 129.

Lab Solutions

1.

a. The following COBOL program calls a VAX BASIC subprogram to add two longword integers. By what passing mechanism does the subprogram expect to receive the passed parameters? *REFERENCE.*

b. By what mechanism is the main program passing the parameters? *DESCRIPTOR.*

c. Copy these programs, and compile, link and run. Explain why the wrong result is obtained.

The subroutine expects to receive parameters by REFERENCE. It wants two addresses of two locations that contain the values 1 and 2. Instead, the main program passes the addresses of two descriptor blocks. The subroutine takes the values in the first word of each descriptor block and adds these values together.

d. What should be the passing mechanism used by the main program? *REFERENCE.*

```
1       *                                              LABSOL1.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID.    LABSOL1.
5       *
6       *   This procedure will invoke the BASIC subprogram
7       *   ADDIT with two integer arguments passed by
8       *   reference.
9       *
10      DATA DIVISION.
11      WORKING-STORAGE SECTION.
12
13          01      I               PIC     9(5)    COMP    VALUE   1.
14          01      J               PIC     9(5)    COMP    VALUE   2.
15
16      PROCEDURE DIVISION.
17      BEGIN.
18      *
19      *   Invoke the BASIC subprogram ADDIT
20          CALL 'ADDIT' USING BY REFERENCE I J.
21          STOP RUN.
```

2.

a.

```
1       *                                                LABSOL2A.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID.    LABSOL2A.
5       *
6       *   This procedure will invoke the VAX-11 Run-Time
7       *   Library procedure LIB$PUT_OUTPUT to output a
8       *   character string to SYS$OUTPUT. The procedure
9       *   is called as a subroutine.
10      *
11      DATA DIVISION.
12      WORKING-STORAGE SECTION.
13
14      01  MSG-STR         PIC X(12)       VALUE 'Hello There!'.
15
16      PROCEDURE DIVISION.
17      BEGIN.
18      *
19      *   Invoke the Run-Time procedure
20          CALL 'LIB$PUT_OUTPUT' USING BY DESCRIPTOR MSG-STR.
21          STOP RUN.
```

b.

```
1       *                                                LABSOL2B.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID.    LABSOL2B.
5       *
6       *   This procedure will invoke the VAX-11 Run-Time
7       *   Library procedure LIB$PUT_OUTPUT to output a
8       *   character string to SYS$OUTPUT. The procedure
9       *   is called as a function.
10      *
11      DATA DIVISION.
12      WORKING-STORAGE SECTION.
13
14      01  MSG-STR          PIC X(12)       VALUE 'Hello There!'.
15      01  STAT             PIC S9(9)       COMP.
16
17      PROCEDURE DIVISION.
18      BEGIN.
19      *
20      *   Invoke the Run-Time procedure
21          CALL 'LIB$PUT_OUTPUT' USING BY DESCRIPTOR MSG-STR
22                                GIVING STAT.
23          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
24          STOP RUN.
```

3.

```
1       *                                                LABSOL3.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID.    LABSOL3.
5       *
6       *   This procedure will:
7       *     1. Clear Event Flag #2 using the system service
8       *           $CLREF
9       *     2. Check Status for success
10      *     3. Print a message if the status was unsuccessful
11      *        or successful
12      *
13      DATA DIVISION.
14      WORKING-STORAGE SECTION.
15
16      01  CLREF-EFN                   PIC  9(9)  COMP     VALUE 2.
17      01  STATUS-RETURN           PIC S9(9)  COMP.
18
19      PROCEDURE DIVISION.
20      BEGIN.
21      *
22      *   Invoke system service to clear event flag #2
23          CALL 'SYS$CLREF' USING  BY VALUE CLREF-EFN
24                                   GIVING STATUS-RETURN.
25      *
26      *   If failure, print message
27          IF STATUS-RETURN IS FAILURE
28               DISPLAY  'Unsuccessful attempt to clear EFN 2.'
29          ELSE
30               DISPLAY  'EFN 2 has been cleared.'.
31          STOP RUN.
```

Synchronizing Processes

File Location

On-line files are provided to help you master this module. The files are located in the directory:

```
DISK$COURSE:[COURSE.V4PROG.COB.SYNC]
```

For your convenience, your system manager may have created the following logical name equivalence:

```
DISK$COURSE:[COURSE.V4PROG.COB.SYNC] = V4PROG$COB$SYNC
```

Three types of files are provided:

1. The Module Programming Examples, which are also listed in this language workbook.
2. Files that you modify to complete the Laboratory Exercises.
3. Solutions to the Laboratory Exercises.

If you wish to modify and/or compile, link, and run any of the programs, first copy the file into your directory.

Example 1

This example illustrates the use of a local event flag.

❶ The number of the allocated event flag is stored in the program in the variable called SIGNAL.

❷ The first argument in the call to $SETIMR is an event flag to be set when the timer expires.

❸ The call to $WAITFR causes the process to be placed in a wait state until the event flag SIGNAL is set.

```
 1       *                                                  LOCFLAG.COB
 2       IDENTIFICATION DIVISION.
 3       *
 4       PROGRAM-ID. LOCFLAG.
 5       *
 6       *   This program allocates a local event flag using a
 7       *   library routine, and sets the event flag using the
 8       *   $SETIMR service.
 9       *
10       DATA DIVISION.
11       WORKING-STORAGE SECTION.
12
13       01 WAIT_SECS        PIC X(9)  VALUE '0 ::25.00'.
14       01 DELAY    PIC 9(18) COMP.
15       01 STAT             PIC S9(9) COMP.
16       01 SIGNAL   PIC 9(9) COMP.
17
18       PROCEDURE DIVISION.
19       BEGIN.
20       *
21       *   Translate ascii time to binary.
22           CALL 'SYS$BINTIM' USING BY DESCRIPTOR WAIT_SECS
23                                   BY REFERENCE DELAY
24                             GIVING STAT.
25           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
26       *
27       *   Allocate the event flag and call it signal
28    (1)    CALL 'LIB$GET_EF' USING SIGNAL
29                             GIVING STAT.
30           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
31       *
32       *   Set timer, specifying event flag signal to be set upon
33       *   expiration of timer
34    (2)    CALL 'SYS$SETIMR' USING BY VALUE SIGNAL
35                                   BY REFERENCE DELAY
36                                   BY VALUE 0 0
37                             GIVING STAT.
38           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
39       *
40    (3)    CALL 'SYS$WAITFR' USING BY VALUE SIGNAL
41                             GIVING STAT.
42           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
43           DISPLAY 'Event flag has been set, so timer has expired'.
44       *
45       *   Deallocate event flag
46           CALL 'LIB$FREE_EF' USING SIGNAL
47                             GIVING STAT.
48           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
49           STOP RUN.
```

```
$ COBOL LOCFLAG
$ LINK LOCFLAG
$ RUN LOCFLAG
Event flag has been set, so timer has expired
$
```

Example 1 Synchronization Using a Local Event Flag

Example 2

This example uses time routines to output the current time and set a timer. Refer to the *VAX/VMS System Services Reference Manual* and the *VAX/VMS Run-Time Library Routines Reference Manual* for detailed descriptions of the routines used.

❶ When specifying a delta time in ASCII format to the $BINTIM service, nonspecified time fields default to zero. If leading fields are omitted, the punctuation must remain. If the number of days is zero, a zero must be specified.

❷ The call to $SETIMR places an entry in the system timer queue.

- The optional argument for specifying an AST routine (the first argument) has been omitted in this call to $SETIMR. It is shown in the following sections. Remember, when optional arguments are omitted from a system service call, the arguments are passed BY VALUE 0.
- The second argument is the address of an array containing the quadword expiration time. The system service expects the address of the quadword; therefore the quadword is passed by reference, which is COBOL's default for numeric arguments.
- The fourth argument is used to assign a timer ID of three to the timer request. (The default ID value is zero.) Timer ID's are used when canceling timer requests.

❸ The $READEF system service indicates whether the event flag is set or clear. The event flag is set when the timer expires; if the flag is clear, the timer has not yet expired.

```
1       *                                                ALARM.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. ALARM.
5       *
6       *   This program illustrates the use of time conversion and
7       *   set timer routines.
8       *
9       DATA DIVISION.
10      WORKING-STORAGE SECTION.
11
12      01  DELAY_INT       PIC S9(18)      COMP    VALUE 0.
13      01  TIMER_ID        PIC S9(9)       COMP    VALUE 3.
14      01  CUR_TIME        PIC X(23).
15  (1) 01  OFF_SET         PIC X(9)        VALUE '0 ::30.00'.
16      01  STAT            PIC S9(9)       COMP.
17      01  CNT             PIC S9(9)       COMP    VALUE 0.
18      01  STATE           PIC S9(9)       COMP.
19      01  SS$_WASCLR      PIC S9(9)       COMP
20                                          VALUE EXTERNAL SS$_WASCLR.
21
22      PROCEDURE DIVISION.
23      BEGIN.
24      *
25      *   Convert time interval from ascii to binary format
26          CALL 'SYS$BINTIM' USING BY DESCRIPTOR OFF_SET
27                                  BY REFERENCE DELAY_INT
28                           GIVING STAT.
29          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
30      *
31      *   Inform user of current time
32          CALL 'LIB$DATE_TIME' USING BY DESCRIPTOR CUR_TIME
33                               GIVING STAT.
34          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
35          DISPLAY 'The time is now: ' CUR_TIME
36      *
37      *   Set timer to expire in 30 seconds
38  (2)     CALL 'SYS$SETIMR' USING BY VALUE 2
39                                  BY REFERENCE DELAY_INT
40                                  BY VALUE 0 TIMER_ID
41                           GIVING STAT.
42          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
43      *
44      *   Count to 1,000,000
45          PERFORM UNTIL CNT = 1000000
46                  ADD 1 TO CNT
47          END-PERFORM.
48
```

Example 2 Placing an Entry in the System Timer Queue (Sheet 1 of 2)

```
   49        *   Check if timer expired
   50     (3)    CALL 'SYS$READEF' USING BY VALUE 2
   51                                  BY REFERENCE STATE
   52                           GIVING STAT.
   53            IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
   54            IF STAT = SS$_WASCLR THEN
   55        *           Cancel timer using id number
   56                    CALL 'SYS$CANTIM' USING BY VALUE TIMER_ID 0
   57                                       GIVING STAT
   58                    IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT
   59                    ELSE
   60                           DISPLAY 'Count completed before timer expired.'
   61            ELSE
   62                    DISPLAY 'Timer expired before count completed.'
   63            STOP RUN.

$ COBOL ALARM
$ LINK ALARM
$ RUN ALARM
The time is now:  3-FEB-1984 11:42:16.61
Count completed before timer expired.
$
```

Example 2 Placing an Entry in the System Timer Queue (Sheet 2 of 2)

Example 3

This example illustrates the use of common event flags for interprocess synchronization. Program RECORD associates to common cluster INFO and waits until program CALCULATE sets the third event flag in INFO.

❶ This example requires two processes in the same group to run the programs RECORD and CALCULATE. This can be done by executing RECORD within a subprocess, or by using two terminals. In both cases, RECORD must run first, then CALCULATE.

❷ The value of 64 for the first argument indicates that cluster number 2 will be associated. Any number from 64 to 95 could have been used.

❸ Notice that the process running RECORD and the process running CALCULATE need not associate INFO as the same cluster number, but the cluster name must be the same.

❹ Event flags in common clusters are relative to the start of the cluster. In this particular example, the event flag that RECORD calls number 66 is the same event flag that CALCULATE calls number 98 (see Figure 1).

```
 1  ❶  *                                                RECORD.COB
 2      IDENTIFICATION DIVISION.
 3      *
 4      PROGRAM-ID. RECORD1.
 5      *
 6      *   This program illustrates interprocess synchronization.
 7      *   It associates to common cluster INFO, then waits for
 8      *   The third event flag in the cluster to be set.
 9      *
10      DATA DIVISION.
11      WORKING-STORAGE SECTION.
12
13      01  CLUSTER_NAME    PIC X(4)        VALUE 'INFO'.
14      01  STAT            PIC S9(9) COMP.
15
16      PROCEDURE DIVISION.
17      BEGIN.
18      *
19      *   Associate to cluster #2 as INFO
20  ❷       CALL 'SYS$ASCEFC' USING BY VALUE 64
21                                  BY DESCRIPTOR CLUSTER_NAME
22                                  BY VALUE 0 0
23                            GIVING STAT.
24          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
25      *
26      *   Wait for third event flag to be set
27          DISPLAY 'Waiting for event flag 66 to be set.'
28  ❹       CALL 'SYS$WAITFR' USING BY VALUE 66
29                            GIVING STAT.
30          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
31      *
32          DISPLAY 'EFN 66 has been set.'.
33          STOP RUN.


$ COBOL RECORD
$ LINK RECORD
$ RUN RECORD
Waiting for event flag 66 to be set.
EFN 66 has been set.
```

Example 3 Process Synchronization Using a Common Event Flag Cluster (Sheet 1 of 2)

```
 1  ❶  *                                                  CALCULATE.COB
 2      IDENTIFICATION DIVISION.
 3      *
 4      PROGRAM-ID. CALCULATE.
 5      *
 6      *   This program will associate to common cluster INFO,
 7      *   then set the third event flag in the cluster,
 8      *   allowing program RECORD to continue.
 9      *
10      DATA DIVISION.
11      WORKING-STORAGE SECTION.
12
13      01  CLUSTER_NAME    PIC X(4)        VALUE 'INFO'.
14      01  STAT            PIC S9(9) COMP.
15
16      PROCEDURE DIVISION.
17      BEGIN.
18      *
19      *   Associate to cluster #3 as INFO
20  ❸       CALL 'SYS$ASCEFC' USING BY VALUE 96
21                                  BY DESCRIPTOR CLUSTER_NAME
22                                  BY VALUE 0 0
23                          GIVING STAT.
24          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
25      *
26      *   Set third event flag
27          DISPLAY 'Setting EFN 98.'.
28  ❹       CALL 'SYS$SETEF' USING BY VALUE 98
29                           GIVING STAT.
30          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
31      *
32          DISPLAY 'finished.'.
33          STOP RUN.
```

```
$ COBOL CALCULATE
$ LINK CALCULATE
$ RUN CALCULATE
Setting EFN 98.
finished.
$
```

Example 3 Process Synchronization Using a Common Event Flag Cluster (Sheet 2 of 2)

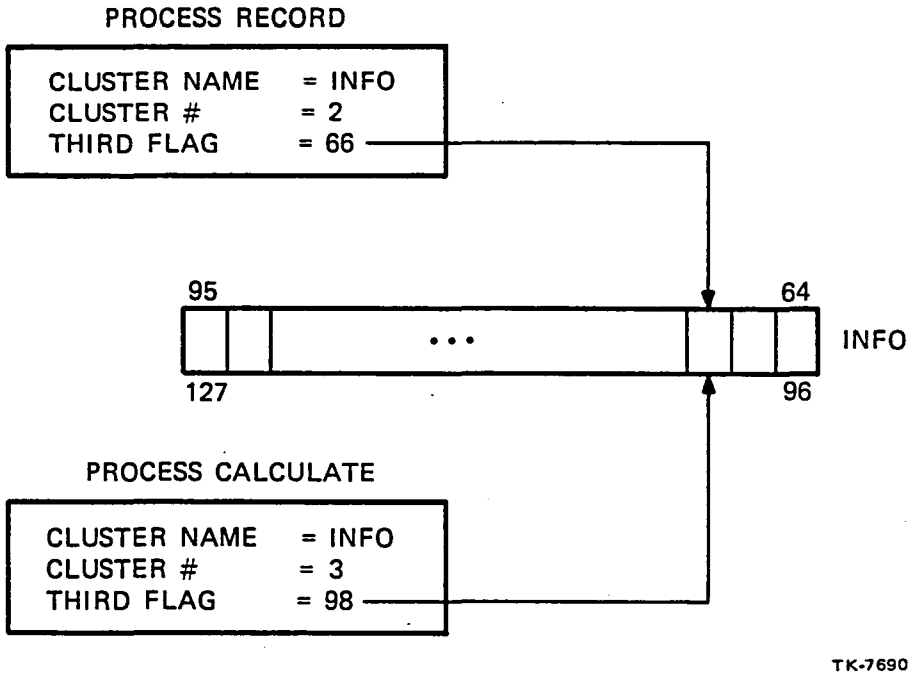

Figure 1 Two Processes Accessing the Same Common Cluster Using Different Cluster Numbers

Example 4

This example illustrates the methods for requesting and declaring an AST procedure. Further examples of using AST procedures appear in other modules.

❶ The third parameter in the $SETIMR call is the address of the entry point of the AST procedure to be executed when the timer expires. Note that the AST procedure must be declared as EXTERNAL.

❷ If the user wants the AST routine to be executed repeatedly, rather than just once, the timer must be reset at the end of the AST routine.

❸ When the AST is delivered, the AST routine interrupts the program and outputs a message.

```
1       *                                                ASTPROC.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. ASTPROC.
5       *
6       *   This program sets a 10 second timer which requests an
7       *   AST.  The main program then performs arithmetic
8       *   operations for the user, interrupted after 10 seconds
9       *   by the timer AST.
10      *
11      DATA DIVISION.
12      WORKING-STORAGE SECTION.
13
14      01  BIN_DELAY        PIC 9(18) COMP.
15      01  DELAY            PIC X(9)  VALUE '0 ::10.00'.
16      01  STAT             PIC S9(9) COMP.
17      01  AST_PROC         PIC S9(9) COMP VALUE EXTERNAL AST_PROC.
18      01  IN-NUM1          PIC S999.
19      01  IN-NUM2          PIC S999.
20      01  NUM1             PIC S9(9) COMP.
21      01  NUM2             PIC S9(9) COMP.
22      01  PRODUCT          PIC S9(9) COMP VALUE IS 0.
23      01  OUT-PRODUCT      PIC -Z(6).
24
25      PROCEDURE DIVISION.
26      BEGIN.
27      *
28      *   Convert delay interval to binary and set timer
29          CALL 'SYS$BINTIM' USING BY DESCRIPTOR DELAY
30                                  BY REFERENCE BIN_DELAY
31                            GIVING STAT.
32  (1)     IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
33          CALL 'SYS$SETIMR' USING BY VALUE 0
34                                  BY REFERENCE BIN_DELAY
35                                  BY VALUE AST_PROC
36                                  BY VALUE 0
37                            GIVING STAT.
38          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
39      *
40      *   Prompt user for two numbers and multiply them
41          PERFORM WITH TEST AFTER UNTIL PRODUCT = 0
42                  DISPLAY 'Enter two integers to be multiplied.'
43                  DISPLAY 'Enter two zeros to quit.'
44                  ACCEPT IN-NUM1
45                  ACCEPT IN-NUM2
46                  MOVE IN-NUM1 TO NUM1
47                  MOVE IN-NUM2 TO NUM2
48                  MULTIPLY NUM1 BY NUM2 GIVING PRODUCT
49                  MOVE PRODUCT TO OUT-PRODUCT
50                  IF PRODUCT IS NOT EQUAL TO 0
51                          DISPLAY 'The product is:' OUT-PRODUCT
52                  END-IF
53          END-PERFORM
54          STOP RUN.
55      END PROGRAM ASTPROC.
```

Example 4 Synchronization Using an AST Routine (Sheet 1 of 2)

```
    56
    57        IDENTIFICATION DIVISION.
    58        *
    59   ❷    PROGRAM-ID. AST_PROC.
    60        *
    61        *   This subprogram is called as an AST procedure.
    62        *   It prints the current time at the terminal.
    63        *
    64        DATA DIVISION.
    65        WORKING-STORAGE SECTION.
    66        01  OUT_STR.
    67            05  MSG           PIC X(18)  VALUE ' The time is now: '.
    68            05  CUR_TIME      PIC X(23)  VALUE SPACES.
    69        01  TERM              PIC X(10)  VALUE 'SYS$OUTPUT'.
    70        01  STAT              PIC S9(9) COMP.
    71        01  IOSB              PIC S9(18) COMP.
    72        01  BRK$C_DEVICE      PIC S9(9) COMP  VALUE 1.
    73
    74        PROCEDURE DIVISION.
    75        BEGIN.
    76            CALL 'LIB$DATE_TIME' USING BY DESCRIPTOR CUR_TIME
    77                                 GIVING STAT.
    78            IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
    79   ❸        CALL 'SYS$BRKTHRUW' USING BY VALUE 1
    80                                       BY DESCRIPTOR OUT_STR
    81                                       'SYS$OUTPUT'
    82                                       BY VALUE BRK$C_DEVICE
    83                                       BY REFERENCE IOSB
    84                                       BY VALUE 0 0 0 0 0 0
    85                               GIVING STAT.
    86            IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
    87            EXIT PROGRAM.
    88        END PROGRAM AST_PROC.
```

```
$ COBOL ASTPROC
$ LINK ASTPROC
$ RUN ASTPROC
Enter two integers to be multiplied.
Enter two zeros to quit.
003
004
The product is:     12
Enter two integers to be multiplied.
Enter two zeros to quit.
 The time is now: 27-FEB-1984 11:26:34.13
012
012
The product is:    144
Enter two integers to be multiplied.
Enter two zeros to quit.
000
000
$
```

Example 4 Synchronization Using an AST Routine (Sheet 2 of 2)

Example 5

Example 5 shows a call to $ENQW to request a lock on a resource name.

❶ The second parameter in the call to $ENQW is the lock mode requested. The lock modes are system-defined symbols in the form of LCK$K_ccMODE, where cc is the two-letter code for the lock mode (listed in the *System Services Reference Manual*).

A data division entry defines a value for the LCK$K_PRMODE symbolic code. To determine the value of a symbolic code:

a. Create the file LCKDEF.MAR:

```
$LCKDEF  GLOBAL
.END
```

b. Issue the DCL commands:

```
$ MACRO/LIST LCKDEF
$ TYPE LCKDEF
```

The symbol table contains the **hexidecimal** values of all LCK$ symbolic codes. The VALUE clause in the data division specifies the decimal equivalent.

❷ STATBLK is a quadword that receives the lock status block information (completion status and lock ID).

❸ The fifth parameter specifies the resource name being locked.

❹ The $DEQ system service is called with the lock ID to dequeue the PR mode lock. (The lock ID is the second longword of STATBLK.)

```
 1       *                                                NEWLOCK.COB
 2       IDENTIFICATION DIVISION.
 3       *
 4       PROGRAM-ID. NEWLOCK.
 5       *
 6       *   SYS$ENQW is used in this program to place a lock on a
 7       *   resource called database.
 8       *
 9       DATA DIVISION.
10       WORKING-STORAGE SECTION.
11
12       01  SHARE_ITEM      PIC X(8)        VALUE 'DATABASE'.
13       01  STATBLK.
14           02 LOCK-STATUS          PIC S9(9) COMP.
15           02 LOCK-ID              PIC S9(9) COMP.
16       01  STAT            PIC S9(9)  COMP.
17       01  LCK$K_PRMODE    PIC S9(9)  COMP VALUE 3.
18
19       PROCEDURE DIVISION.
20       BEGIN.
21           DISPLAY 'Enqueueing lock...'.
22       (1)     CALL 'SYS$ENQW' USING BY VALUE 0 LCK$K_PRMODE
23       (2)                           BY REFERENCE STATBLK
24                                     BY VALUE 0
25       (3)                           BY DESCRIPTOR SHARE_ITEM
26                                     BY VALUE 0 0 0 0 0 0
27                             GIVING STAT.
28           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
29           IF LOCK-STATUS IS FAILURE CALL 'LIB$STOP'
30                             USING BY VALUE LOCK-STATUS.
31           DISPLAY 'Lock has been granted.'.
32       (4)     CALL 'SYS$DEQ' USING BY VALUE LOCK-ID 0 0 0
33                            GIVING STAT.
34           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
35           DISPLAY 'Lock has been dequeued.'.
36           STOP RUN.
```

```
$ COBOL NEWLOCK
$ LINK NEWLOCK
$ RUN NEWLOCK
Enqueueing lock...
Lock has been granted.
Lock has been dequeued.
$
```

Example 5 Requesting a Resource Lock

Example 6

This example shows lock mode conversion and uses a blocking AST routine. The two programs are executed by two processes sharing a resource called DATABASE. One process (process A) acts as a background process, requiring frequent protected write (PW) access to the resource. A PW lock, however, is not compatible with many other lock modes. Rather than repeatedly calling $ENQ and $DEQ each time it needs access, process A places a PW lock and specifies a blocking AST.

Whenever another process (process B, in this case) requests a lock that is blocked by process A's PW lock, the blocking AST is delivered to process A. At which time, process A converts its lock to a null (NL) lock. The following notes refer to Example 6 and Figure 2.

❶ A PW lock is requested on DATABASE.

- The $ENQW system service is used (instead of $ENQ) because the lock must be received before the program can continue.
- Notice that the address of the blocking AST routine is the ninth parameter (BLKAST), NOT the seventh parameter (ASTASR).

❷ Process B expresses interest in the resource by requesting a NL lock on the resource name. This should be done in the initialization section of any application.

❸ Process B enqueues a conversion request.

- Process B's request for conversion to a PR lock is placed in the conversion queue because it is not compatible with the PW lock.
- This request causes the blocking AST to be delivered to process A.

❹ Execution of the AST routine causes HAVELOCK to be set to zero, and causes process A to convert its PW lock to NL mode.

❺ Once process A's lock is converted to NL mode, process B's conversion to PR is allowed.

```
1       *                                                WRITELOCK.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. WRITELOCK.
5       *
6       *           This program uses the blocking AST feature of the lock
7       *   manager.
8       *
9       DATA DIVISION.
10      WORKING-STORAGE SECTION.
11
12      01  SHARE_ITEM      PIC X(8)        VALUE 'DATABASE'.
13      01  STATBLK.
14          05  SBLK_1      PIC S9(9) COMP.
15          05  SBLK_2      PIC S9(9) COMP.
16      01  HAVELOCK        PIC S9(9) COMP  EXTERNAL.
17      01  DONE            PIC S9(9) COMP.
18      01  BLOCKING        PIC S9(9) COMP  VALUE EXTERNAL BLOCKING.
19      01  LCK$K_PWMODE    PIC S9(9) COMP  VALUE 4.
20      01  LCK$K_NLMODE    PIC S9(9) COMP  VALUE 0.
21      01  LCK$M_CONVERT   PIC S9(9) COMP  VALUE 2.
22      01  STAT            PIC S9(9) COMP.
23
24      PROCEDURE DIVISION.
25      BEGIN.
26          SET DONE TO SUCCESS.
27      *
28      *   Place a PW lock on DATABASE
29          DISPLAY 'Placing a PW lock on DATABASE...'.
30      (1) CALL 'SYS$ENQW' USING BY VALUE 0 LCK$K_PWMODE
31                                BY REFERENCE STATBLK
32                                BY VALUE 0
33                                BY DESCRIPTOR SHARE_ITEM
34                                BY VALUE 0 0 0 BLOCKING 0 0
35                        GIVING STAT.
36          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
37          IF SBLK_1 IS FAILURE CALL 'LIB$STOP' USING BY VALUE SBLK_1.
38          SET HAVELOCK TO SUCCESS.
39          PERFORM WRITE-OPS UNTIL
40                  HAVELOCK IS FAILURE AND DONE IS SUCCESS.
41      *
42      *   Convert PW lock to NL
43      (4) CALL 'SYS$ENQW' USING BY VALUE 0 LCK$K_NLMODE
44                                BY REFERENCE STATBLK
45                                BY VALUE LCK$M_CONVERT
46                                BY DESCRIPTOR SHARE_ITEM
47                                BY VALUE 0 0 0 0 0 0
48                        GIVING STAT.
49          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
50          IF SBLK_1 IS FAILURE CALL 'LIB$STOP' USING BY VALUE SBLK_1.
51          DISPLAY 'LOCK converted from PW to NL mode.'.
52          STOP RUN.
53      WRITE-OPS.
54      *
55      *   Operations which require write access to the
56      *   resource could be performed here.
57      *           Set DONE to success when finished
58      END PROGRAM WRITELOCK.
```

Example 6 Resource Lock Conversion and Blocking AST Routines (Sheet 1 of 3)

```
59
60          IDENTIFICATION DIVISION.
61          PROGRAM-ID. BLOCKING.
62          *
63          *   This is used as a blocking AST routine.  It tells the main
64          *   program (via a flag) to release its lock, which is blocking
65          *   another lock.
66          *
67          DATA DIVISION.
68          WORKING-STORAGE SECTION.
69          01  HAVELOCK        PIC S9(9) COMP EXTERNAL.
70          PROCEDURE DIVISION.
71          BEGIN.
72              SET HAVELOCK TO FAILURE.
73              DISPLAY 'Our lock is blocking another lock.'
74              EXIT PROGRAM.
75          END PROGRAM BLOCKING.


 1          *                                                READLOCK.COB
 2          IDENTIFICATION DIVISION.
 3          PROGRAM-ID. READLOCK.
 4          *
 5          *   This program enqueues a null lock on the resource
 6          *   resource DATABASE, which is later converted to a
 7          *   protected read lock.
 8          *
 9          DATA DIVISION.
10          WORKING-STORAGE SECTION.
11          01  SHARE_ITEM      PIC X(8)        VALUE 'DATABASE'.
12          01  STATBLK.
13              05  SBLK_1      PIC S9(9) COMP.
14              05  SBLK_2      PIC S9(9) COMP.
15          01  LCK$K_NLMODE    PIC S9(9) COMP  VALUE 0.
16          01  LCK$K_PRMODE    PIC S9(9) COMP  VALUE 3.
17          01  LCK$M_CONVERT   PIC S9(9) COMP  VALUE 2.
18          01  STAT            PIC S9(9) COMP.
19
20          PROCEDURE DIVISION.
21          BEGIN.
22          *
23          *   Place a null lock on DATABASE
24              DISPLAY 'Placing a NL lock on the  resource...'.
25     (2)      CALL 'SYS$ENQW' USING BY VALUE 0 LCK$K_NLMODE
26                                    BY REFERENCE STATBLK
27                                    BY VALUE 0
28                                    BY DESCRIPTOR SHARE_ITEM
29                                    BY VALUE 0 0 0 0 0 0
30                               GIVING STAT.
31              IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
32              IF SBLK_1 IS FAILURE CALL 'LIB$STOP' USING BY VALUE SBLK_1.
33          *
34          *   Operations not requiring access to the resource
35          *       could be performed here.
36          *           Convert null lock to PR
37              DISPLAY 'Converting lock from NL to PR mode...'.
38     (3)      CALL 'SYS$ENQW' USING BY VALUE 0 LCK$K_PRMODE
39                                    BY REFERENCE STATBLK
40                                    BY VALUE LCK$M_CONVERT
41                                    BY DESCRIPTOR SHARE_ITEM
42                                    BY VALUE 0 0 0 0 0 0
43                               GIVING STAT.
44              IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
45              IF SBLK_1 IS FAILURE CALL 'LIB$STOP' USING BY VALUE SBLK_1.
46          *
47          *   Operations which require reading of the resource
48          *   could be performed here.
49     (5)      DISPLAY 'Finished.'.
50              STOP RUN.
```

Example 6 Resource Lock Conversion and Blocking AST Routines (Sheet 2 of 3)

```
$ ! PROCESS A
$ !
$ COBOL WRITELOCK
$ LINK WRITELOCK
$ RUN WRITELOCK
Placing a PW lock on DATABASE...
Our lock is blocking another lock.
LOCK converted from PW to NL mode.
```

```
$ ! PROCESS B
$ !
$ COBOL READLOCK
$ LINK READLOCK
$ RUN READLOCK
Placing a NL lock on the  resource...
Converting lock from NL to PR mode...
Finished.
$
```

Example 6 Resource Lock Conversion and Blocking AST Routines (Sheet 3 of 3)

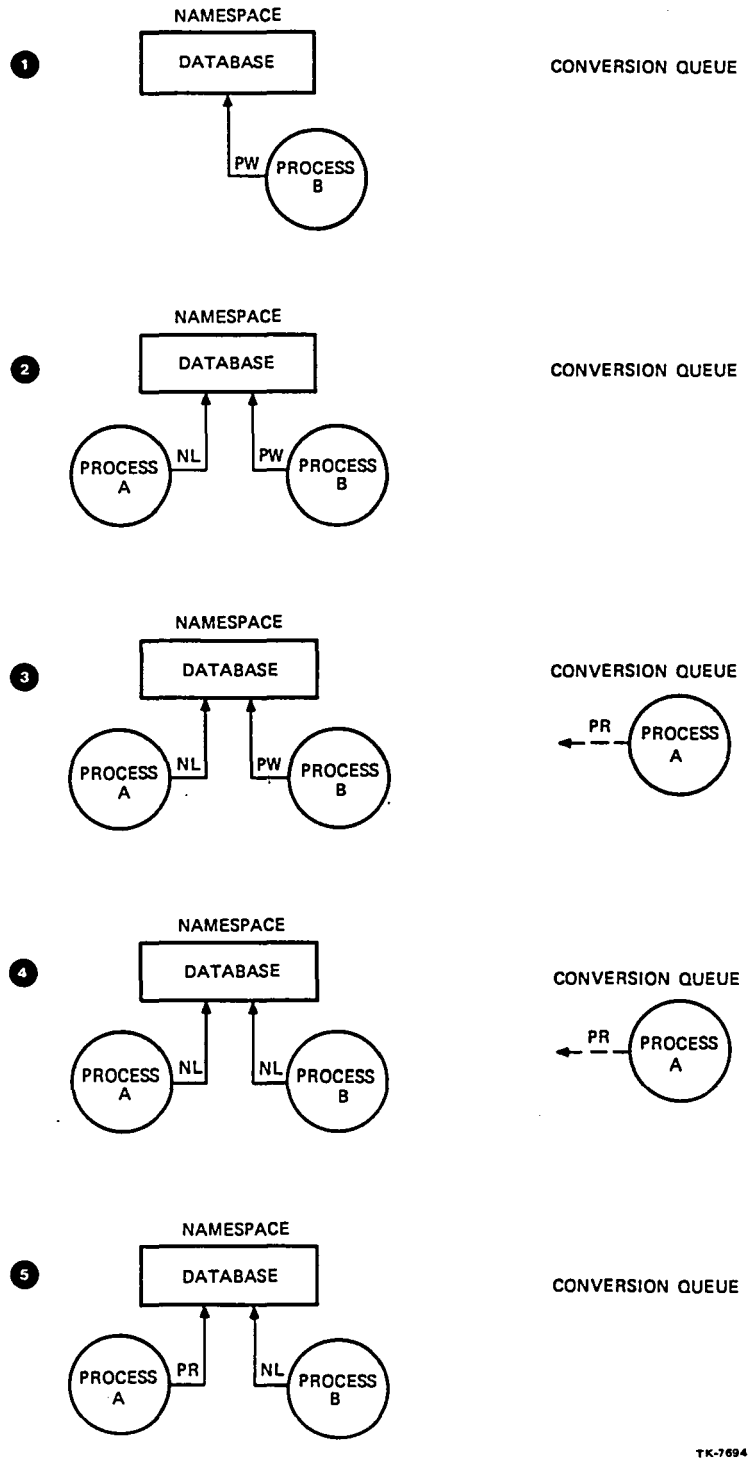

Figure 2 Creating and Converting Locks

Lab Exercises

1. The program provided (LAB1.COB) was designed to read a number (N) from the terminal and then output a message to the terminal twice, with an N-second delay between the writes. All the necessary variables and data have been defined, but three system service calls have been omitted. You are asked to supply:

   a. A call to $BINTIM to convert the ASCII delay time to binary. DELAY_INTERVAL is a quadword that will receive the converted time.

   b. A call to $SETIMR to schedule the setting of event flag number 1 after a delay of N seconds.

   c. A call to $WAITFR to cause the program to wait until event flag number 1 is set.

   Code these calls, then compile, link and run the program.

Provided File

```
*                                                                    LAB1.COB
PROGRAM
IDENTIFICATION DIVISION.
*
PROGRAM-ID.   LAB1.
*
*      This procedure:
*      1. Prompts for a delay time, in seconds
*      2. Prints a message twice with the input delay
*
DATA DIVISION.
WORKING-STORAGE SECTION.

01 ASCII_INTERVAL.
   02 A                                    PIC X(4)        VALUE '0 ::'.
   02 B                                    PIC X(2).
   02 C                                    PIC X(3)        VALUE '.00'.

01 DELAY_INTERVAL                          PIC S9(18) COMP VALUE 0.
01 MY-NAME                                 PIC X(8)        VALUE 'Jone Doe'.
01 ASCII_SECONDS                            PIC X(2).
01 STAT                                    PIC S9(9)       COMP.

PROCEDURE DIVISION.
BEGIN.
*
*      Get delay time
       DISPLAY 'Enter a delay time from 1 to 59 sec.: '
               WITH NO ADVANCING.
       ACCEPT  ASCII_SECONDS.
*
*      Convert ASCII delay to binary
       MOVE    ASCII_SECONDS TO B.
*
*      Call to SYS$BINTIM here ******************************
*
       IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
*
*      Print first message
       DISPLAY MY_NAME.
*
*      Wait the delay time
*
*      Call to SYS$SETIMR here ******************************
*
       IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
*
*      Call to SYS$WAITFR here ******************************
*
       IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
*
*      Print second message
       DISPLAY MY_NAME.
       STOP RUN.
```

2. Using the skeleton program provided (LAB2.COB), write a main program that:

   a. Allocates a local event flag from process-wide pool using the Run-Time Library routine LIB$GET_EF.

   b. Clears the event flag using $CLREF in case the program is run again.

   c. Converts an ASCII time interval of five seconds to binary time using $BINTIM.

   d. Prints the current time obtained using $ASCTIM.

   e. Uses $SETIMR to deliver an AST in five seconds and set the flag. The flag is passed as a parameter to the AST routine.

   f. Uses the AST routine provided in the skeleton program to set the flag.

   g. Tests the status of the event flag using $READEF. If the status is SS$_WASCLR, then test again. If the status is SS$_WASSET, then continue. If any other status is returned, call LIB$STOP.

   h. Prints the current time obtained using $ASCTIM.

   i. Ends the program.

   Be sure to call each system service as a function, and test the return status after each call.

Provided File

```
*                                                    LAB2.COB
IDENTIFICATION DIVISION.
*
PROGRAM-ID.   LAB2.
*
*      This procedure will:
*       1. Allocate an event flag from process-wide pool
*          and clear it
*       2. Print the current time of day
*       3. Issue a timer request to deliver an AST and
*          set the local event flag after five seconds have
*          elapsed
*       4. Wait for the local event flag used in the
*          AST procedure to be set
*       5. Print the current time.
*
DATA DIVISION.
WORKING-STORAGE SECTION.

01   ASCII_INTERVAL              PIC X(9)        VALUE '0 ::05.00'.
01   ASCII_TIME                  PIC X(23)       VALUE SPACES.
01   BINARY_INTERVAL              PIC S9(18)COMP         VALUE 0.
01   AST                           PIC 9(9)        COMP   VALUE EXTERNAL AST.
01   FLAG                          PIC 9(9) COMP.
01   STAT                         PIC S9(9) COMP.
01   STATE                        PIC S9(9) COMP.
01   SS$_WASCLR        PIC S9(9) COMP VALUE EXTERNAL SS$_WASCLR.
01   SS$_WASSET        PIC S9(9) COMP VALUE EXTERNAL SS$_WASSET.
```

(Sheet 1 of 2)

```
PROCEDURE DIVISION.
BEGIN.
*
*       Allocate event flag from process-wide pool
*
*       Clear event flag
*
*       Convert ASCII to binary time
*
*       Print the current time
*
*       Deliver an AST and set the event flag in 5 seconds
*
*       Read status of event flag and loop until set
*
*       Print the current time
        STOP RUN.


END PROGRAM LAB2.

IDENTIFICATION DIVISION.
*
PROGRAM-ID.       AST.
*
DATA DIVISION.
WORKING-STORAGE SECTION.

01  STAT                         PIC S9(9) COMP.

LINKAGE SECTION.

01  FLAG                         PIC 9(9) COMP.

PROCEDURE DIVISION USING FLAG.
BEGIN.
*
*       Set the event flag
        CALL 'SYS$SETEF'USING BY VALUE FLAG
                        GIVING STAT.
        IF STAT IS FAILURE CALL 'LIB$STOP'USING BY VALUE STAT.
        EXIT PROGRAM.
END PROGRAM AST.
```

(Sheet 2 of 2)

3. Using the program provided (LAB3.COB), write another program (LABSOL3.COB) to create a deadlock situation when both programs are run simultaneously.

   Program LAB3.COB does the following:

   - Places a protected write lock on resource 1.
   - Waits 10 seconds.
   - Places a protected write lock on resource 2.
   - Checks the lock status block from the last $ENQ system service to determine if the lock on resource 2 was granted, or if deadlock occurred.
   - If deadlock occurred, issues a message and unlocks resource 1.

   Copy LAB3.COB to a program called LABSOL3.COB and modify it to do the following:

   - Place a protected write lock on resource 2.
   - Wait 10 seconds.
   - Place a protected write lock on resource 1.
   - Check the lock status block from the last $ENQ system service to determine if the lock on resource 1 was granted or if deadlock occurred.
   - If deadlock occurred, issue a message and unlock resource 2.

   Log in under the same username at two different terminals. Run LAB3 and LABSOL3 simultaneously, and observe the deadlock situation.

Provided File

```
*                                                    LAB3.COB
IDENTIFICATION DIVISION.
*
PROGRAM-ID.  LAB3.
*
*       This program, in conjunction with LABSOL3, creates
*       a deadlock situation. This program attempts to
*       lock two resources for its use in a specific order.
*       LABSOL3 attempts to lock the same resources
*       in the opposite order. This cross locking
*       creates a deadlock situation when the two
*       programs are run simultaneously.
*
DATA DIVISION.
WORKING-STORAGE SECTION.

01      RES_1                PIC X(9)  VALUE 'MY_RES_1'
01      RES_2                PIC X(9)  VALUE 'MY_RES_2'
01      STBLK1.
        02 LOCK-STATUS-1     PIC S9(9) COMP.
        02 LOCK-ID-1         PIC S9(9) COMP.
01      STBLK2.
        02 LOCK-STATUS-2     PIC S9(9) COMP.
        02 LOCK-ID-2         PIC S9(9) COMP.
01      STAT                 PIC S9(9) COMP.
01      LCK$K_PWMODE         PIC S9(9) COMP  VALUE 4.
01      PERIOD               PIC X(9)  VALUE '0 ::10.00'
01      ABSPER               PIC S9(18) COMP VALUE 0.
01      SS$_DEADLOCK         PIC S9(9) COMP VALUE EXTERNAL
        SS$_DEADLOCK.

PROCEDURE DIVISION.
BEGIN.
*
* Tellthe user that this process has started.
      DISPLAY     '
      DISPLAY     ''
      DISPLAY     ' LAB3 has started.'.
```

(Sheet 1 of 3)

```
*
*   Lock the first resource desired. Check to see
*   if the service was accepted. If the lock was
*   accepted, tell the user.
      CALL 'SYS$ENQW' USING BY VALUE 0 LCK$K_PWMODE
                            BY REFERENCE STBLK1
                            BY VALUE 0
                            BY DESCRIPTOR RES_1
                            BY VALUE 0 0 0 0 0 0
                               GIVING STAT.
      IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
      DISPLAY ' ... Resource 1 granted'.
*
*   Wait a specified period of time before locking
*   the second. Absolute time must be used.
      CALL 'SYS$BINTIM' USING BY DESCRIPTOR PERIOD
                            BY REFERENCE ABSPER
                               GIVING STAT.
      IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
      CALL 'SYS$SETIMR' USING BY VALUE 4
                            BY REFERENCE ABSPER
                            BY VALUE 0 0
                               GIVING STAT.
    IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
    DISPLAY    ' ... Wait 10 seconds.'.
    CALL 'SYS$WAITFR' USING BY VALUE 4
                               GIVING STAT.
    IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
*
*   Lock the second resource if possible. Wait until
*   it is.
      DISPLAY    ' ... Queue a PW lock on resource 2.'.
      CALL 'SYS$ENQ' USING BY VALUE 7 LCK$K_PWMODE
                            BY REFERENCE STBLK2
                            BY VALUE 0
                            BY DESCRIPTOR RES_2
                            BY VALUE 0 0 0 0 0 0
                               GIVING STAT.
    IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
    CALL 'SYS$WAITFR' USING BY VALUE 7
                               GIVING STAT.
    IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
```

(Sheet 2 of 3)

```
*
*   Check the most recent status block returned from the
*   last SYS$ENQ system service. If a deadlock situation
*   exists, tell the user and unlock the first resource
*   (which has already been granted) and then exit this
*   process.
      IF LOCK-STATUS-2 IS NOT EQUAL TO SS$_DEADLOCK THEN
            DISPLAY ' ... Hold resource 2 for 10
            seconds.'
            CALL 'SYS$SETIMR' USING BY VALUE 9
                                    BY REFERENCE
                                    ABSPER
                                    BY VALUE 0 0
                        GIVING STAT
            IF STAT IS FAILURE
              CALL 'LIB$STOP' USING BY VALUE STAT END-IF
            CALL 'SYS$WAITFR' USING BY VALUE 9
                        GIVING STAT
            IF STAT IS FAILURE
              CALL 'LIB$STOP' USING BY VALUE STAT END-IF
      ELSE
            DISPLAY ' ... I am the victim of the DEADLOCK!'
            CALL 'SYS$DEQ' USING BY VALUE LOCK-ID-1 0 0 0
                        GIVING STAT
            IF STAT IS FAILURE
              CALL 'LIB$STOP' USING BY VALUE STAT END-IF
   END-IF.
   DISPLAY '  LAB3 has finished.'
   DISPLAY '  '
   DISPLAY '  '
   STOP RUN.
```

(Sheet 3 of 3)

Lab Solutions

1.

```
1       *                                                LABSOL1.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID.    LABSOL1.
5       *
6       *   This procedure:
7       *    1. Prompts for a delay time, in seconds
8       *    2. Prints a message twice with the input delay
9       *
10      DATA DIVISION.
11      WORKING-STORAGE SECTION.
12
13      01  ASCII_INTERVAL.
14          02      A                       PIC X(4)        VALUE '0 ::'.
15          02  B                   PIC X(2).
16          02  C                   PIC X(3)        VALUE '.00'.
17
18      01  DELAY_INTERVAL          PIC S9(18) COMP VALUE 0.
19      01  MY-NAME                 PIC X(8)        VALUE 'Jane Doe'.
20      01  ASCII_SECONDS               PIC X(2).
21      01  STAT                    PIC S9(9)       COMP.
22
23      PROCEDURE DIVISION.
24      BEGIN.
25      *
26      *   Get delay time
27          DISPLAY  'Enter a delay time from 1 to 59 sec.: '
28                   WITH NO ADVANCING.
29          ACCEPT   ASCII_SECONDS.
30      *
31      *   Convert ASCII delay to binary
32                MOVE     ASCII_SECONDS TO B.
33          CALL 'SYS$BINTIM' USING BY DESCRIPTOR ASCII_INTERVAL
34                                  BY REFERENCE DELAY_INTERVAL
35                                GIVING STAT.
36          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
37      *
38      *   Print first message
39          DISPLAY MY_NAME.
40      *
41      *   Wait the delay time
42          CALL 'SYS$SETIMR' USING BY VALUE 1
43                                  BY REFERENCE DELAY_INTERVAL
44                                  BY VALUE 0
45                                  BY VALUE 0
46                              GIVING STAT.
47          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
48          CALL 'SYS$WAITFR' USING BY VALUE 1
49                              GIVING STAT.
50          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
51      *
52      *   Print second message
53          DISPLAY MY_NAME.
54          STOP RUN.
```

2.

```
1       *                                                LABSOL2.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID.    LABSOL2.
5       *
6       *   This procedure will:
7       *     1. Allocate an event flag from process-wide pool
8       *        and clear it
9       *     2. Print the current time of day
10      *     3. Issue a timer request to deliver an AST and
11      *        set the local event flag after five seconds have
12      *        elapsed
13      *     4. Wait for the local event flag used in the
14      *        AST procedure to be set
15      *     5. Print the current time.
16      *
17      DATA DIVISION.
18      WORKING-STORAGE SECTION.
19
20      01  ASCII_INTERVAL     PIC X(9)         VALUE '0 ::05.00'.
21      01  ASCII_TIME         PIC X(23)        VALUE SPACES.
22      01  BINARY_INTERVAL    PIC S9(18) COMP      VALUE 0.
23      01  AST                PIC 9(9)   COMP VALUE EXTERNAL AST.
24      01  FLAG               PIC 9(9)   COMP.
25      01  STAT               PIC S9(9) COMP.
26      01  STATE              PIC S9(9) COMP.
27      01  SS$_WASCLR         PIC S9(9) COMP  VALUE EXTERNAL SS$_WASCLR.
28      01  SS$_WASSET         PIC S9(9) COMP  VALUE EXTERNAL SS$_WASSET.
29
30      PROCEDURE DIVISION.
31      BEGIN.
32      *
33      *   Allocate event flag from process-wide pool
34              CALL 'LIB$GET_EF' USING BY REFERENCE FLAG.
35      *
36      *   Clear event flag
37          CALL 'SYS$CLREF' USING BY VALUE FLAG GIVING STAT.
38          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
39      *
40      *   Convert ASCII to binary time
41          CALL 'SYS$BINTIM' USING BY DESCRIPTOR ASCII_INTERVAL
42                                          BY REFERENCE BINARY_INTERVAL
43                                    GIVING STAT.
44          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
45      *
46      *   Print the current time
47          CALL 'SYS$ASCTIM' USING BY VALUE 0
48                                          BY DESCRIPTOR ASCII_TIME
49                                          BY VALUE 0 0
50                                    GIVING STAT.
51          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
52          DISPLAY 'The current time is ' ASCII_TIME.
53      *
54      *   Deliver an AST and set the event flag in 5 seconds
55          CALL 'SYS$SETIMR' USING BY VALUE 0
56                                          BY REFERENCE BINARY_INTERVAL
57                                          BY VALUE AST
58                                    BY REFERENCE FLAG
59                                    GIVING STAT.
60          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
```

(Sheet 1 of 2)

```
61        *
62        *   Read status of event flag
63            PERFORM READ-EF WITH TEST AFTER
64                    UNTIL STAT IS EQUAL TO SS$_WASSET.
65        *
66        *   Print the current time
67            CALL 'SYS$ASCTIM' USING BY VALUE 0
68                                    BY DESCRIPTOR ASCII-TIME
69                                    BY VALUE 0 0
70                                 GIVING STAT.
71            IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
72            DISPLAY 'The current time is '  ASCII_TIME.
73            STOP RUN.
74
75        READ-EF.
76            CALL 'SYS$READEF' USING BY VALUE FLAG
77                                    BY REFERENCE STATE
78                                 GIVING STAT.
79            IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
80        *
81        *   Loop until flag is set
82            IF STAT IS EQUAL TO SS$_WASCLR
83               DISPLAY 'Looping'.
84
85        END PROGRAM LABSOL2.
86
87
88
89        IDENTIFICATION DIVISION.
90        *
91        PROGRAM-ID.         AST.
92        *
93        DATA DIVISION.
94        WORKING-STORAGE SECTION.
95
96        01  STAT                    PIC S9(9) COMP.
97
98        LINKAGE SECTION.
99
100       01  FLAG                    PIC  9(9) COMP.
101
102       PROCEDURE DIVISION USING FLAG.
103       BEGIN.
104       *
105       *   Set the event flag
106           CALL 'SYS$SETEF' USING BY VALUE FLAG
107                                GIVING STAT.
108           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
109           EXIT PROGRAM.
110       END PROGRAM AST.
```

(Sheet 2 of 2)

3.

```
1       *                                                LABSOL3.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID.    LABSOL3.
5       *
6       *   This program, in conjunction with LAB3, creates
7       *   a deadlock situtation.  This program attempts to
8       *   lock two resources for its use in a specific order.
9       *   LAB3 attempts to the lock the same resources
10      *   in the opposite order.  This cross locking
11      *   creates a deadlock situation when the two
12      *   programs are run simultaneously.
13      *
14      DATA DIVISION.
15      WORKING-STORAGE SECTION.
16
17      01  RES_1                    PIC X(9)   VALUE 'MY_RES_1'.
18      01  RES_2                    PIC X(9)   VALUE 'MY_RES_2'.
19      01  STBLK1.
20          02 LOCK-STATUS-1         PIC S9(9) COMP.
21          02 LOCK-ID-1             PIC S9(9) COMP.
22      01  STBLK2.
23          02 LOCK-STATUS-2         PIC S9(9) COMP.
24          02 LOCK-ID-2             PIC S9(9) COMP.
25      01  STAT                     PIC S9(9) COMP.
26      01  LCK$K_PWMODE             PIC S9(9) COMP  VALUE 4.
27      01          PERIOD                  PIC X(9)    VALUE '0 ::10.00'.
28      01          ABSPER                  PIC S9(18) COMP VALUE 0.
29      01  SS$_DEADLOCK             PIC S9(9)  COMP VALUE EXTERNAL
30                  SS$_DEADLOCK.
31
32      PROCEDURE DIVISION.
33      BEGIN.
34      *
35      *  Tell the user that this process has started.
36          DISPLAY          '  '
37          DISPLAY          '  '
38          DISPLAY          '  LABSOL3 has started.'.
39      *
40      *  Lock the first resource desired.  Check to see
41      *  if the service was accepted.  If the lock was
42      *  accepted, tell the user.
43          CALL 'SYS$ENQW' USING BY VALUE 0 LCK$K_PWMODE
44                                BY REFERENCE STBLK2
45                                BY VALUE 0
46                                BY DESCRIPTOR RES_2
47                                BY VALUE 0 0 0 0 0 0
48                           GIVING STAT.
49          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
50          IF LOCK-STATUS-2 IS FAILURE CALL 'LIB$STOP'
51                           USING BY VALUE LOCK-STATUS-2.
52          DISPLAY '  ... Resource 2 granted'.
53      *
54      *  Wait a specified period of time before locking
55      *  the second.  Absolute time must be used.
56          CALL 'SYS$BINTIM' USING BY DESCRIPTOR PERIOD
57                                  BY REFERENCE ABSPER
58                                GIVING STAT.
59          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
60          CALL 'SYS$SETIMR' USING BY VALUE 4
61                                  BY REFERENCE ABSPER
62                                  BY VALUE 0 0
63                                GIVING STAT.
64          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
65          DISPLAY '  ... Wait 10 seconds.'.
66          CALL 'SYS$WAITFR' USING BY VALUE 4
67                                GIVING STAT.
68          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
```

(Sheet 1 of 2)

```
69      *
70      *  Lock the second resource if possible.  Wait until
71      *  it is.
72          DISPLAY  '  ... Queue a PW lock on resource 1.'.
73          CALL 'SYS$ENQ' USING BY VALUE 7 LCK$K_PWMODE
74                               BY REFERENCE STBLK1
75                               BY VALUE 0
76                               BY DESCRIPTOR RES_1
77                               BY VALUE 0 0 0 0 0 0
78                         GIVING STAT.
79          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
80          CALL 'SYS$WAITFR' USING BY VALUE 7
81                                  GIVING STAT.
82          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
83      *
84      *   Check the most recent status block returned from the
85      *   last SYS$ENQ system service.  If a deadlock situation
86      *   exists, tell the user and unlock the first resource
87      *   (which has already been granted) and then exit this
88      *   process.
89          IF  LOCK-STATUS-1 IS NOT EQUAL TO SS$_DEADLOCK  THEN
90                  IF LOCK-STATUS-1 IS FAILURE CALL 'LIB$STOP'
91                          USING BY VALUE LOCK-STATUS-1
92                  END-IF
93                  DISPLAY  '  ... Hold resource 1 for 10 seconds.'
94                  CALL 'SYS$SETIMR' USING BY VALUE 9
95                                          BY REFERENCE ABSPER
96                                          BY VALUE 0 0
97                            GIVING STAT
98                  IF STAT IS FAILURE
99                      CALL 'LIB$STOP' USING BY VALUE STAT
100                 END-IF
101                 CALL 'SYS$WAITFR' USING BY VALUE 9
102                                     GIVING STAT
103                 IF STAT IS FAILURE
104                     CALL 'LIB$STOP' USING BY VALUE STAT
105                 END-IF
106         ELSE
107                         DISPLAY   '... I am the victim of the DEADLOCK!'
108                 CALL 'SYS$DEQ' USING BY VALUE LOCK-ID-2 0 0 0
109                                GIVING STAT
110                 IF STAT IS FAILURE
111                     CALL 'LIB$STOP' USING BY VALUE STAT
112                 END-IF
113         END-IF.
114         DISPLAY '  LABSOL3 has finished.'
115         DISPLAY '  '
116         DISPLAY '  '
117         STOP RUN.
```

(Sheet 2 of 2)

Accessing Devices

File Locations

On-line files are provided to help you master this module. The files are located in the directory:

```
DISK$COURSE:[COURSE.V4PROG.COB.DEVC]
```

For your convenience, your system manager may have created the following logical name equivalence:

```
DISK$COURSE:[COURSE.V4PROG.COB.DEVC] = V4PROG$COB$DEVC
```

Two types of files are provided:

1. The Module Programming Examples, which are also listed in this language workbook.
2. Solutions to the Laboratory Exercises

If you wish to modify and/or compile, link, and run any of the programs, first copy the file into your directory.

Example 1

Example 1 performs output to a terminal through the $QIOW system service.

❶ TERM__CHAN receives the channel number from the $ASSIGN system service.

The process permanent logical name SYS$COMMAND is assigned to your terminal when you log in. The $ASSIGN system service translates the logical name to the actual device name.

❷ $QIO and $QIOW accept the CHAN argument by immediate value, unlike $ASSIGN, which supplies it by reference. Note the use of %VAL in the call to $QIOW, but not in the call to $ASSIGN.

The function IO$__WRITEVBLK requires values for parameters P1, P2, and P4.

- P1 is the starting address of the buffer containing the message. So TEXT__STRING is passed by reference.
- P2 is the number of bytes to be written to the terminal. A 21 is passed, since it is the length of the message string.
- P4 is the carriage control specifier; a 32 indicates single space carriage control.

A $QIOW is issued, to ensure that the output operation will be completed before the program terminates. Change the $QIOW to a $QIO and see what happens.

```
 1       *                                                QIOW.COB
 2       IDENTIFICATION DIVISION.
 3       *
 4       PROGRAM-ID. QIOW.
 5       *
 6       *   This program illustrates the use of the $QIO
 7       *   system service to perform synchronous I/O to
 8       *   a terminal.
 9       *
10       DATA DIVISION.
11       WORKING-STORAGE SECTION.
12
13       01  TEXT_STRING     PIC X(21)
14                           VALUE 'This is from a $QIOW.'.
15       01  TERM_CHAN       PIC 9(9)  COMP.
16       01  MY_TERMINAL     PIC X(10)       VALUE 'SYS$OUTPUT'.
17       01  IO$_WRITEVBLK   PIC S9(9) COMP
18                           VALUE EXTERNAL IO$_WRITEVBLK.
19       01  STAT            PIC S9(9) COMP.
20       01  IOSB.
21           02      IOSB1           PIC S9(9) COMP.
22           02      IOSB2           PIC S9(9) COMP.
23
24       PROCEDURE DIVISION.
25       BEGIN.
26       *
27       *   Assign the channel number
28  (1)      CALL 'SYS$ASSIGN' USING BY DESCRIPTOR MY_TERMINAL
29                                   BY REFERENCE TERM_CHAN
30                                   BY VALUE 0 0
31                                   GIVING STAT.
32           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
33       *
34       *   Output the message twice
35           PERFORM  2 TIMES
36  (2)         CALL 'SYS$QIOW' USING BY VALUE 0 TERM_CHAN
37                                               IO$_WRITEVBLK
38                                    BY REFERENCE IOSB
39                                    BY VALUE 0 0
40                                    BY REFERENCE TEXT_STRING
41                                    BY VALUE 21 0 32 0 0
42                               GIVING STAT
43               IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT
44                   END-IF
45               IF IOSB1 IS FAILURE CALL 'LIB$STOP' USING BY VALUE IOSB1
46               END-IF
47           END-PERFORM.
48           STOP RUN.
```

```
$ COBOL QIOW
$ LINK QIOW
$ RUN QIOW
This is from a $QIOW.
This is from a $QIOW.
$
```

Example 1 Synchronous I/O

Example 2

Example 2, NOECHO.COB, shows how to combine the function codes and function code modifiers. A $QIOW with the IO$__READVBLK function code can be used to read from a terminal. If you do not want the user input to echo on the terminal screen, then you must use the IO$M__NOECHO modifier. NOECHO.COB reads input from the terminal without echoing the user input.

❶ The EXTERNAL clauses define the function code IO$__READVBLK and modifier IO$M-__NOECHO.

❷ The function code and modifier are ORed and then passed BY VALUE to the $QIOW in the third parameter.

```
 1       *                                              NOECHO.COB
 2       IDENTIFICATION DIVISION.
 3       *
 4       PROGRAM-ID.    NOECHO.
 5       *
 6       *   This procedure shows how to combine function codes
 7       *   and modifiers.  It does a read from th terminal
 8       *   without echoing the input characters.
 9       *
10       DATA DIVISION.
11       WORKING-STORAGE SECTION.
12
13       01  TT1_NAME          PIC X(11)       VALUE 'SYS$COMMAND'.
14       01  TT1_CHAN          PIC 9(9)  COMP  VALUE 0.
15     ( 01  IO$_READVBLK      PIC S9(9) COMP  VALUE EXTERNAL IO$_READVBLK.
16   1 ) 01  IO$M_NOECHO       PIC S9(9) COMP  VALUE EXTERNAL IO$M_NOECHO.
17       01  STATUS_BLOCK.
18           02 IOSB           OCCURS 4 TIMES.
19            03 TT1_IOSB     PIC S9(4) COMP.
20       01  STAT              PIC S9(9) COMP  VALUE 0.
21       01  FUNC              PIC S9(9) COMP.
22       01  BUF_1             PIC X(20) VALUE SPACES.
23       01  PROMPT            PIC X(16) VALUE 'Enter your name '.
24
25       PROCEDURE DIVISION.
26       BEGIN.
27       *
28       *   Assign channel to the terminal
29           CALL 'SYS$ASSIGN' USING BY DESCRIPTOR TT1_NAME
30                                   BY REFERENCE TT1_CHAN
31                                   BY VALUE 0 0
32                                GIVING STAT.
33           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
34       *
35       *   Issue read to terminal
36           DISPLAY PROMPT WITH NO ADVANCING.
37           COMPUTE FUNC = IO$_READVBLK + IO$M_NOECHO
38   2       CALL 'SYS$QIOW' USING BY VALUE 1 TT1_CHAN FUNC
39                                 BY REFERENCE STATUS_BLOCK
40                            BY VALUE 0 0
41                                 BY REFERENCE BUF_1
42                                 BY VALUE 20 0 0 0 0
43                              GIVING STAT.
44           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
45           IF TT1_IOSB(1) IS FAILURE
46                   CALL 'LIB$STOP' USING BY VALUE TT1_IOSB(1).
47       *
48       *   Write data back to terminal
49           DISPLAY SPACES.
50           DISPLAY 'Hi ', BUF_1(1:TT1_IOSB(2)).
51           STOP RUN.


$ COBOL NOECHO
$ LINK NOECHO
$ RUN NOECHO
Enter your name
Hi NANCY
$
```

Example 2 Combining I/O Function Code and Modifier

Example 3

Example 3 uses all three methods for determining when I/O has been performed.

❶ When VMS completes its output to the terminal, it does the following:

- Sets local event flag 1
- Fills the IOSB, named STATUS__BLOCK
- Runs the AST procedure, AST__PROC

❷ A COBOL procedure receives all arguments by reference. Therefore, the ASTPRM argument STATUS__BLOCK must be passed to $QIO by reference.

❸ The $SYNCH system service is used to determine I/O completion. It tests both the event flag and the I/O status block to determine I/O completion.

```
1       *                                                    ASYNCH.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. ASYNCH.
5       *
6       *   This program illustrates various ways to determine
7       *   $QIO completion.  It also illustrates the use of an
8       *   IOSB to obtain information about the I/O operation.
9       *
10      DATA DIVISION.
11      WORKING-STORAGE SECTION.
12
13      01  TEXT_STRING     PIC X(29)
14                               VALUE 'This was written by the $QIO.'.
15      01  TERM_CHAN       PIC 9(9)  COMP.
16      01  MY_TERMINAL     PIC X(10)       VALUE 'SYS$OUTPUT'.
17      01  STAT            PIC S9(9) COMP.
18      01  STATUS-BLOCK.
19          05      IOSB-STATUS             PIC S9(4) COMP.
20          05      IOSB-COUNT              PIC S9(4) COMP.
21          05      IOSB-DEV-INFO           PIC S9(4) COMP.
22      01  AST_PROC        PIC S9(9) COMP  VALUE EXTERNAL AST_PROC.
23      01  IO$_WRITEVBLK   PIC S9(9) COMP  VALUE EXTERNAL
24                          IO$_WRITEVBLK.
25      01  STRING_LENGTH   PIC 9(9)  COMP  VALUE 29.
26      01  FORMS_CONTROL   PIC 9(9)  COMP  VALUE 32.
27
28      PROCEDURE DIVISION.
29      BEGIN.
30      *
31      *   Assign an I/O channel
32          CALL 'SYS$ASSIGN' USING BY DESCRIPTOR MY_TERMINAL
33                                  BY REFERENCE TERM_CHAN
34                                  BY VALUE 0 0
35                                  GIVING STAT.
36          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
37      *
38      *   Queue the I/O
39  (1)     CALL 'SYS$QIO' USING BY VALUE 1
40                                       TERM_CHAN
41                                       IO$_WRITEVBLK
42  (2)                          BY REFERENCE STATUS_BLOCK
43                               BY VALUE AST_PROC
44                               BY REFERENCE STATUS_BLOCK
45                                            TEXT_STRING
46                               BY VALUE STRING_LENGTH 0
47                                       FORMS_CONTROL 0 0
48                          GIVING STAT.
49          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
```

Example 3 Asynchronous I/O (Sheet 1 of 2)

```
50          *
51          *   Wait for the I/O Operation to complete
52      (3)     CALL 'SYS$SYNCH' USING BY VALUE 1
53                                     BY REFERENCE STATUS_BLOCK
54                                GIVING STAT.
55          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
56          IF IOSB_STATUS IS FAILURE
57                  CALL 'LIB$STOP' USING BY VALUE IOSB_STATUS.
58          DISPLAY 'The I/O operation and AST procedure are done.'.
59          STOP RUN.
60      END PROGRAM ASYNCH.
61
62
63      IDENTIFICATION DIVISION.
64      *
65      PROGRAM-ID. AST_PROC.
66      *
67      *   This subprogram is called as an AST procedure.
68      *   it uses the AST parameter passed to it by the
69      *   $QIO as an IOSB to determine how may characters
70      *   were written to the terminal.
71      *
72      DATA DIVISION.
73      WORKING-STORAGE SECTION.
74
75      01  IOSB-DISP.
76          05      DISP-STATUS             PIC ZZZ9.
77          05      DISP-COUNT              PIC ZZZ9.
78
79      LINKAGE SECTION.
80
81      (2) 01  WRITE-STATUS.
82          05      IOSB-STATUS             PIC S9(4) COMP.
83          05      IOSB-COUNT              PIC S9(4) COMP.
84          05      IOSB-DEV-INFO           PIC S9(4) COMP.
85
86      (2) PROCEDURE DIVISION USING WRITE-STATUS.
87      BEGIN.
88      *
89      *   Write byte count and I/O status to the terminal
90          MOVE IOSB-COUNT TO DISP-COUNT.
91          MOVE IOSB-STATUS TO DISP-STATUS.
92          DISPLAY 'The number of characters output is' DISP-COUNT.
93          DISPLAY 'The I/O completion status is     ' DISP-STATUS.
94          EXIT PROGRAM.
95      END PROGRAM AST_PROC.
```

```
$ COBOL ASYNCH
$ LINK ASYNCH
$ RUN ASYNCH
This was written by the $QIO.
The number of characters output is  32
The I/O completion status is        1
The I/O operation and AST procedure are done.
$
$
```

Example 3 Asynchronous I/O (Sheet 2 of 2)

Example 4

Example 4 uses the $MOUNT system service to mount a foreign tape. It then reads EBCDIC data from the tape, converts the data to ASCII, and displays it.

❶ The item list for $MOUNT consists of four item descriptors, followed by a zero longword.

The second word in the item descriptor defines the value of the item codes. To determine the value of an item code:

a. Create the file MNTDEF.MAR:

```
$MNTDEF     GLOBAL
```

b. Issue the DCL commands:

```
$ MACRO/LIST MNTDEF
$ TYPE MNTDEF
```

The symbol table contains the hexidecimal values of all MNT$ symbolic codes. The decimal equivalents are used in the VALUE clauses.

The POINTER VALUE REFERENCE clause in each item descriptor stores the address of the data names in the item list.

❷ The tape is mounted as foreign, as specified by the flags argument. Reading data from a tape is a logical I/O operation, which normally requires the LOG_IO privilege. Logical I/O operations can be performed to a device that is mounted foreign without the LOG_IO privilege.

❸ One parameter, the item list, is required for the call to $MOUNT. The $MOUNT system service implicitly allocates the tape drive, so a call to $ALLOC is not needed.

❹ The open statement prepares the file for I/O operations. A channel is assigned between the tape drive and the process.

❺ The CLOSE statement deassigns the channel.

❻ The $DISMOU system service dismounts and unloads the tape. You can dismount a tape without unloading it using the optional flags argument. The $DISMOU system service also deallocates the tape drive.

```
1       *                                                FOREIGN.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. FOREIGN.
5       *
6       *   This program illustrates reading from a foriegn
7       *   magnetic tape containing EBCDIC data.  The tape
8       *   must be written with fixed length records that
9       *   do not span blocks.
10      *
11      ENVIRONMENT DIVISION.
12      INPUT-OUTPUT SECTION.
13      FILE-CONTROL.
14
15          SELECT FOREIGN_TAPE ASSIGN TO MY_TAPE.
16
17      DATA DIVISION.
18      FILE SECTION.
19      *
20      *   This program assumes a record length of 80 with 1
21      *   record per block
22      FD  FOREIGN_TAPE
23                  LABEL RECORDS ARE STANDARD
24                  BLOCK CONTAINS 80 CHARACTERS
25                  FILE STATUS IS TAPE_STATUS.
26      01  DATA_LINE          PIC X(80).
27
28      WORKING-STORAGE SECTION.
29  (1) 01  MNT_LIST.
30        02  ITEM-DEVICE-NAME.
31          03          PIC S9(4)       COMP VALUE 4.
32          03          PIC S9(4)       COMP VALUE 1.
33          03    POINTER VALUE REFERENCE TAPE_UNIT.
34          03    PIC S9(9)   COMP VALUE 0.
35        02  ITEM-BLOCK-SIZE.
36          03          PIC S9(4)       COMP VALUE 4.
37          03          PIC S9(4)       COMP VALUE 8.
38          03    POINTER VALUE REFERENCE BLOCKS.
39          03    PIC S9(9)   COMP VALUE 0.
40        02  ITEM-RECORD-SIZE.
41          03          PIC S9(4)       COMP VALUE 4.
42          03          PIC S9(4)       COMP VALUE 16.
43          03    POINTER VALUE REFERENCE RECERDS.
44          03    PIC S9(9)   COMP VALUE 0.
45  (2)   02  ITEM-FLAGS.
46          03          PIC S9(4)       COMP VALUE 4.
47          03          PIC S9(4)       COMP VALUE 4.
48          03    POINTER VALUE REFERENCE FLAGS.
49          03    PIC S9(9)   COMP VALUE 0.
50        02  TERMINATOR-ENTRY       PIC S9(9) COMP  VALUE 0.
51      01  TAPE_UNIT          PIC X(4)        VALUE 'MTA0'.
52      01  BLOCKS             PIC S9(9) COMP  VALUE 80.
53      01  RECERDS            PIC S9(9) COMP  VALUE 80.
54      01  FLAGS              PIC S9(9) COMP  VALUE 1.
55      01  TAPE_STATUS        PIC XX.
56      01  STAT               PIC S9(9) COMP.
57
```

Example 4 Performing I/O on Foreign Devices (Sheet 1 of 2)

```
58        PROCEDURE DIVISION.
59        BEGIN.
60        *
61        *   Mount the tape
62     3      CALL 'SYS$MOUNT' USING MNT_LIST
63                              GIVING STAT.
64            IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
65        *
66        *   Open the tape file
67     4      OPEN INPUT FOREIGN_TAPE.
68        *
69        *   Read and convert each record until end-of-file
70            PERFORM UNTIL TAPE_STATUS = '13'
71                    READ FOREIGN_TAPE AT END GO TO CLEAN-UP
72                    END-READ
73        *
74        *           Convert from EBCIDIC to ASCII
75                    CALL 'LIB$TRA_EBC_ASC' USING
76                                            BY DESCRIPTOR DATA_LINE
77                                                          DATA_LINE
78                                           GIVING STAT
79        *
80        *           Code to process ASCII records goes here
81                    DISPLAY DATA_LINE
82            END-PERFORM.
83        CLEAN-UP.
84        *
85        *   Close the file, dismount the tape
86     5      CLOSE FOREIGN_TAPE.
87     6      CALL 'SYS$DISMOU' USING BY DESCRIPTOR TAPE_UNIT
88                                    BY VALUE 0
89                             GIVING STAT.
90            IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
91            STOP RUN.

$ COBOL FOREIGN
$ LINK FOREIGN
$ ASSIGN MTA0: MY_TAPE
$ RUN FOREIGN
This is a test of the $MOUNT system service
These 3 data lines are records read from a
foreign tape & translated from EBCDIC to ASCII
$
```

Example 4 Performing I/O on Foreign Devices (Sheet 2 of 2)

Lab Exercises

1. Modify Example 2, NOECHO.COB, so that the prompt string (Enter your name) is initiated from the $QIOW. The resulting program should function like NOECHO.COB. Keep the following questions in mind:

   - What $QIOW function code is needed to issue a prompt from within a read statement?
   - What function code modifier is needed to disable echoing?

2. Write a program that uses $QIOW and does the following:

   - Assigns a channel to the terminal
   - Prompts the user for a text string
   - Allows the user up to 10 seconds to complete the input
   - Determines if the user completed the input
   - Types the string if the input was completed
   - Displays an error message if the input was not completed

   Keep the following questions in mind:

   - What function code modifier is needed to enable the $QIOW timeout feature?
   - What error code is returned if the $QIOW times out?

The error code returned on timeout (SS$__TIMEOUT) is described in terminal driver information in Appendix A of the *VAX/VMS File Structured Devices Reference Manual.*

3. Write a program that does the following:
   - Assigns a channel to the terminal
   - Issues a $QIO to the terminal that:
     - Reads a message from the terminal
     - Specifies an AST completion routine and passes the channel number as the AST parameter. The AST should:
       a. Use the $GETDVIW system service to get the device name
       b. Write the device name to the terminal
   - Issues a $QIOW to write the message back to the terminal

Lab Solutions

1.

```
1       *                                                LABSOL1.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID.    LABSOL1.
5       *
6       *   This program uses a $QIOW system service call to
7       *   display a prompt at the terminal, and to accept
8       *   user input without echo.
9       *
10      DATA DIVISION.
11      WORKING-STORAGE SECTION.
12
13      01  TT_CHAN          PIC 9(9)  COMP.
14      01  MY_TERMINAL      PIC X(11)       VALUE 'SYS$COMMAND'.
15      01  IN_STR           PIC X(20)       VALUE SPACES.
16      01  PROMPT           PIC X(16)       VALUE 'Enter your name '.
17      01  FUNC_CODE        PIC S9(9) COMP.
18      01  IO$_READPROMPT   PIC S9(9) COMP  VALUE EXTERNAL IO$_READPROMPT.
19      01  IO$M_NOECHO      PIC S9(9) COMP  VALUE EXTERNAL IO$M_NOECHO.
20      01  STATUS_BLOCK.
21          03 IOSB_STAT     PIC S9(4) COMP.
22          03 IOSB_COUNT    PIC S9(4) COMP.
23          03 IOSB_INFO     PIC S9(9) COMP.
24      01  STAT             PIC S9(9) COMP.
25
26      PROCEDURE DIVISION.
27      BEGIN.
28      *
29      *   Assign a channel number to the terminal
30      *
31          CALL 'SYS$ASSIGN' USING BY DESCRIPTOR MY_TERMINAL
32                                  BY REFERENCE TT_CHAN
33                                  BY VALUE 0 0
34                                  GIVING STAT.
35          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
36          COMPUTE FUNC_CODE = IO$_READPROMPT + IO$M_NOECHO
37      *
38      *   Issue QIOW
39      *
40          CALL 'SYS$QIOW' USING BY VALUE 1 TT_CHAN
41                                       FUNC_CODE
42                                BY REFERENCE STATUS_BLOCK
43                                BY VALUE 0 0
44                                BY REFERENCE IN_STR
45                                BY VALUE 20 0 0
46                                BY REFERENCE PROMPT
47                                BY VALUE 16
48                                GIVING STAT.
49          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
50          IF IOSB_STAT IS FAILURE
51                  CALL 'LIB$STOP' USING BY VALUE IOSB_STAT.
52          DISPLAY SPACES
53          DISPLAY 'Hi ' IN_STR(1:IOSB_COUNT)
54          STOP RUN.
```

2.

```
1       *                                                    LABSOL2.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID.    LABSOL2.
5       *
6       *   This procedure:
7       *    1. Prompts for a text string
8       *    2. Prints a message if no input within 10 seconds
9       *    3. Types the string if input was completed.
10      *
11      DATA DIVISION.
12      WORKING-STORAGE SECTION.
13
14      01  STAT              PIC S9(9) COMP.
15      01  TERMNAL           PIC X(2)        VALUE 'TT'.
16      01  TERM_CHAN         PIC 9(9)  COMP  VALUE 0.
17      01  IO$_READPROMPT    PIC S9(9) COMP  VALUE EXTERNAL IO$_READPROMPT.
18      01  IO$M_TIMED        PIC S9(9) COMP  VALUE EXTERNAL IO$M_TIMED.
19      01  SS$_TIMEOUT       PIC S9(9) COMP  VALUE EXTERNAL SS$_TIMEOUT.
20      01  IO_FUNC           PIC S9(9) COMP  VALUE 0.
21      01  STATUS_BLOCK.
22          02 TEXT_IOSB     OCCURS 4 TIMES.
23                  03  IOSB         PIC S9(4)  COMP.
24
25      01  TEXT_BUF                 PIC X(80) VALUE SPACES.
26      01  PROMPT_MSG               PIC X(21) VALUE 'Enter a text string:'.
27
28      PROCEDURE DIVISION.
29      BEGIN.
30      *
31      *   Assign a channel to TERMINAL
32          CALL 'SYS$ASSIGN' USING BY DESCRIPTOR TERMNAL
33                                  BY REFERENCE TERM_CHAN
34                                  BY VALUE 0 0
35                                GIVING STAT.
36          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
37      *
38      *   Prompt for the text string
39          ADD IO$_READPROMPT IO$M_TIMED GIVING IO_FUNC.
40          CALL 'SYS$QIOW' USING  BY VALUE 1 TERM_CHAN IO_FUNC
41                                 BY REFERENCE IOSB(1)
42                                 BY VALUE 0 0
43                                 BY REFERENCE TEXT_BUF
44                                 BY VALUE 80 10 0
45                                 BY REFERENCE PROMPT_MSG
46                                 BY VALUE 21
47                               GIVING STAT.
48          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
49      *
50      *   If I/O time-out, notify user
51          IF IOSB(1) IS EQUAL TO SS$_TIMEOUT THEN
52                  DISPLAY SPACES
53                  DISPLAY 'No message in 10 seconds.'
54          ELSE
55                  IF IOSB(1) IS FAILURE THEN
56                          CALL 'LIB$STOP' USING BY VALUE IOSB(1)
57                  ELSE
58                          DISPLAY TEXT_BUF(1:IOSB(2)).
59          STOP RUN.
```

3.

```
1       *                                                LABSOL3.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID.    LABSOL3.
5       *
6       *   This procedure:
7       *    1. Assigns a channel to a terminal using a
8       *       logical name
9       *    2. Issues an asynchronous read to it (specifying
10      *       an AST)
11      *    3. When the read completes, writes the data back
12      *       to the terminal
13      *
14      DATA DIVISION.
15      WORKING-STORAGE SECTION.
16
17      01  TT1_NAME                PIC X(6)        VALUE 'TERM_1'.
18      01  TT1_CHAN                PIC 9(9)  COMP  VALUE 0.
19      01  IO$_READPROMPT  PIC S9(9) COMP  VALUE EXTERNAL IO$_READPROMPT.
20      01  IO$_WRITEVBLK   PIC S9(9) COMP  VALUE EXTERNAL IO$_WRITEVBLK.
21      01  STATUS_BLOCK.
22          02 IOSB         OCCURS 4 TIMES.
23           03 TT1_IOSB    PIC S9(4) COMP.
24      01  ASTPROC                 PIC 9(9)  COMP  VALUE EXTERNAL ASTPROC.
25      01  BUF_1           PIC X(80)       VALUE SPACES.
26      01  PROMPT_BUF              PIC X(12)       VALUE 'ENTER TEXT: '.
27      01  STAT            PIC S9(9) COMP  VALUE 0.
28
29      PROCEDURE DIVISION.
30      BEGIN.
31      *
32      *   Assign channel to the device
33          CALL 'SYS$ASSIGN' USING BY DESCRIPTOR TT1_NAME
34                                  BY REFERENCE TT1_CHAN
35                                  BY VALUE 0 0
36                              GIVING STAT.
37          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
38      *
39      *   Issue asynchronous read
40          CALL 'SYS$QIO' USING BY VALUE 1 TT1_CHAN
41                                  IO$_READPROMPT
42                               BY REFERENCE STATUS_BLOCK
43                          BY VALUE ASTPROC
44                               BY REFERENCE TT1_CHAN BUF_1
45                               BY VALUE 80 0 0
46                               BY REFERENCE  PROMPT_BUF
47                          BY VALUE 12
48                            GIVING STAT.
49          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
50      *
51      *   Wait for event flag
52          CALL 'SYS$WAITFR' USING BY VALUE 1
53          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
```

(Sheet 1 of 2)

```
54      *
55      *   Write data back to terminal
56          CALL 'SYS$QIOW' USING BY VALUE 2
57                                            TT1_CHAN
58                                            IO$_WRITEVBLK
59                                        BY REFERENCE STATUS_BLOCK
60                                        BY VALUE 0 0
61                                        BY REFERENCE BUF_1
62                                                BY VALUE TT1_IOSB(2)
63                                                 0 32 0 0
64                                        GIVING STAT.
65          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
66          IF TT1_IOSB(1) IS FAILURE
67                  CALL 'LIB$STOP' USING BY VALUE TT1_IOSB(1).
68      *
69      END PROGRAM LABSOL3.
70
71
72
73      IDENTIFICATION DIVISION.
74      *
75      PROGRAM-ID. ASTPROC.
76      *
77      *   This AST is invoked by input from the terminal.
78      *   The only parameter is the channel number.
79      *   The AST procedure:
80      *    1. Obtains the full device name of the device
81      *       (using $GETDVI)
82      *    2. Writes the device name to it
83      *
84      DATA DIVISION.
85
86      WORKING-STORAGE SECTION.
87      01  ITEMDES.
88          02  BUFF_LEN     PIC S9(4) COMP  VALUE 64.
89          02  ITEM_CODE    PIC S9(4) COMP VALUE 32.
90          02  BUFF_ADR.
91          03 POINTER VALUE REFERENCE DEVICE_NAME.
92          02  STR_LEN              PIC S9(9) COMP  VALUE 0.
93          02  TERMIN               PIC S9(9) COMP  VALUE 0.
94      01  IO$_WRITEVBLK   PIC S9(9) COMP  VALUE EXTERNAL IO$_WRITEVBLK.
95      01  DVI$_DEVNAM             PIC S9(9) COMP  VALUE 32.
96      01  STAT            PIC S9(9) COMP    VALUE 0.
97      01  DEVICE_NAME     PIC X(64) VALUE SPACES.
98      01  IOSB.
99          02  IOSB1               PIC S9(9) COMP.
100         02  IOSB2               PIC S9(9) COMP.
101
102     LINKAGE SECTION.
103     01  TERM_CHAN       PIC 9(9)        COMP.
104
105     PROCEDURE DIVISION USING TERM_CHAN.
106     BEGIN.
107     *
108     *   Get the device name
109     *   MOVE DVI$_DEVNAM TO ITEM_CODE.
110         CALL 'SYS$GETDVIW' USING BY VALUE 3 TERM_CHAN 0
111                                           BY REFERENCE BUFF_LEN IOSB
112                                           BY VALUE 0 0 0
113                                    GIVING STAT.
114         IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
115         IF IOSB1 IS FAILURE CALL 'LIB$STOP' USING BY VALUE IOSB1.
116         DISPLAY 'The device name is ' DEVICE_NAME.
117                 EXIT PROGRAM.
118         END PROGRAM ASTPROC.
```

(Sheet 2 of 2)

Communicating with Other Processes

File Locations

On-line files are provided to help you master this module. The files are located in the directory:

```
DISK$COURSE:[COURSE.V4PROG.COB.COMU]
```

For your convenience, your system manager may have created the following logical name equivalence:

```
DISK$COURSE:[COURSE.V4PROG.COB.COMU] = V4PROG$COB$COMU
```

Two types of files are provided:

1. The Module Programming Examples, which are also listed in this language workbook.
2. Solutions to the Laboratory Exercises.

If you wish to modify and/or compile, link, and run any of the programs, first copy the file into your directory.

Example 1

This example shows how to create a temporary mailbox, read from a mailbox, and write to a mailbox. The programs MAILS and MAILR must be run from two processes in the same job. To run the programs, log in at one terminal, run one program as a subprocess and run the other interactively.

❶ Because both MAILS and MAILR use $CREMBX to create a temporary mailbox with the logical name FOZZIE_BEAR (if no such mailbox exists) and assign a channel to it, the programs can be run in any order. If MAILS had used $CREMBX and MAILR had used $ASSIGN, for example, it would be necessary to run MAILS first.

❷ Because $QIOW is used for input and output rather than $QIO, both MAILS and MAILR wait for I/O to complete before advancing to the next program statement.

❸ The P1 argument of $QIOW is the address of a buffer of ASCII text, not the address of an ASCII string descriptor. Therefore, MESAGE and IN_PUT are declared as character data and passed by reference.

```
1       *                                                MAILS.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. MAILS.
5       *
6       *   This program will write to a mailbox with the
7       *   logical name FOZZIE_BEAR.  Since this is a
8       *   synchronous operation, the write will not
9       *   complete until the process MAILR reads the
10      *   mailbox.
11      *
12      ENVIRONMENT DIVISION.
13      INPUT-OUTPUT SECTION.
14      FILE-CONTROL.
15
16          SELECT MAILBOX_NAME ASSIGN TO 'FOZZIE_BEAR'.
17
18      DATA DIVISION.
19      FILE SECTION.
20      FD  MAILBOX_NAME.
21      01  MESAGE          PIC X(18).
22
23      WORKING-STORAGE SECTION.
24      01  MESSAGE_TEXT    PIC  X(18)
25                          VALUE 'This is a message.'.
26      01  IO$_WRITEVBLK   PIC S9(9) COMP
27                          VALUE EXTERNAL IO$_WRITEVBLK.
28      01  CHANNEL         PIC  9(4) COMP.
29      01  STAT            PIC S9(9) COMP.
30      01  IOSB.
31          02  IOSB1       PIC S9(9) COMP.
32          02  IOSB2       PIC S9(9) COMP.
33
34      PROCEDURE DIVISION.
35      BEGIN.
36      *
37      *   Create mailbox and/or assign a channel to it
38  (1)     CALL 'SYS$CREMBX' USING BY VALUE 0
39                                  BY REFERENCE CHANNEL
40                                  BY VALUE 0 0 0 0
41                                  BY DESCRIPTOR 'FOZZIE_BEAR'
42                              GIVING STAT.
43          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
44      *
45      *   Write a message to the mailbox
46          MOVE MESSAGE_TEXT TO MESAGE.
47  (2)     CALL 'SYS$QIOW' USING BY VALUE 2 CHANNEL IO$_WRITEVBLK
48                                BY REFERENCE IOSB
49                                BY VALUE 0 0
50  (3)                           BY REFERENCE MESAGE
51                                BY VALUE 18 0 0 0 0
52                          GIVING STAT.
53          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
54          IF IOSB1 IS FAILURE CALL 'LIB$STOP' USING BY VALUE IOSB1.
55      *
56      *   Notify user
57          DISPLAY 'MAILS has completed mailbox write.'
58          STOP RUN.
```

Example 1 Communicating With Mailboxes (Sheet 1 of 3)

```
1       *                                                MAILR.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. MAILR.
5       *
6       *   This process will read the mailbox created by
7       *   MAILS.  As soon as it reads the mailbox MAILS
8       *   can continue.
9       *
10      ENVIRONMENT DIVISION.
11      INPUT-OUTPUT SECTION.
12      FILE-CONTROL.
13
14          SELECT MAILBOX ASSIGN TO 'FOZZIE_BEAR'.
15
16      DATA DIVISION.
17      FILE SECTION.
18      FD  MAILBOX.
19      01  MESAGE          PIC X(18).
20
21      WORKING-STORAGE SECTION.
22      01  MAILBOX_NAME    PIC X(11)
23                           VALUE 'FOZZIE_BEAR'.
24      01  IO$_READVBLK    PIC S9(9) COMP
25                           VALUE EXTERNAL IO$_READVBLK.
26      01  CHANNEL         PIC  9(4) COMP.
27      01  IN_PUT          PIC X(30).
28      01  STAT            PIC S9(9) COMP.
29      01  IOSB.
30          02  IOSB1       PIC S9(9) COMP.
31          02  IOSB2       PIC S9(9) COMP.
32
33      PROCEDURE DIVISION.
34      BEGIN.
35      *
36      *   Create mailbox and/or assign a channel to it
37  (1)     CALL 'SYS$CREMBX' USING BY VALUE 0
38                                  BY REFERENCE CHANNEL
39                                  BY VALUE 0 0 0 0
40                                  BY DESCRIPTOR MAILBOX_NAME
41                            GIVING STAT.
42          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
43      *
44      *   Write a message to the mailbox
45  (2)     CALL 'SYS$QIOW' USING BY VALUE 2 CHANNEL IO$_READVBLK
46                                BY REFERENCE IOSB
47                                BY VALUE 0 0
48  (3)                           BY REFERENCE IN_PUT
49                                BY VALUE 30 0 0 0 0
50                          GIVING STAT.
51          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
52          IF IOSB1 IS FAILURE CALL 'LIB$STOP' USING BY VALUE IOSB1.
53      *
54      *   Notify user
55          DISPLAY IN_PUT.
56          STOP RUN.
```

Example 1 Communicating With Mailboxes (Sheet 2 of 3)

```
$ ! RUN FROM ONE TERMINAL
$ COBOL MAILS
$ LINK MAILS
$ RUN MAILS
MAILS has completed mailbox write.


$ ! RUN FROM ANOTHER TERMINAL
$ COBOL MAILR
$ LINK MAILR
$ RUN MAILR
This is a message.
$
```

Example 1 Communicating With Mailboxes (Sheet 3 of 3)

Example 2

Example 2 shows how to create a global pagefile section and how two processes can use it to access the same data. One process executes the program PAGEFIL1, which creates and writes to a global pagefile section. PAGEFIL1 then waits for a second process to update the section. The second process executes PAGEFIL2, which maps and updates the pagefile section. Because PAGEFIL2 maps to the temporary global pagefile section created in PAGEFIL1, PAGEFIL1 must be run first. The two processes coordinate their activity through common event flags.

❶ Associate to a common event flag cluster to coordinate activity. The processes must be in the same UIC group.

❷ The $CRMPSC system service creates and maps a global pagefile section.

The starting and ending process virtual addresses of the section are placed in MY__ADR. The output argument SYS__ADR receives the starting and ending system virtual addresses. The flag SEC$M__GLOBAL requests a global section. The flag SEC$M__WRT indicates that the pages should be writable as well as readable. The SEC$M__DZRO flag requests pages filled with zeros. The SEC$M__PAGFIL flag requests a temporary pagefile section.

❸ Data is written to the pagefile section.

❹ PAGEFIL2 maps the existing section as writable by specifying the SEC$M__WRT flag.

❺ The $CRMPSC system service maps pages starting at page boundaries. Since all named COMMON blocks in COBOL are longword (not page) aligned, you must ensure that IARRAY starts on a page boundary. The PSECT construct in the options file for the LINKER accomplishes this. PAGEFIL1 and PAGEFIL2 are linked with the options file.

```
1       *                                                PAGEFIL1.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. PAGEFIL1.
5       *
6       *   This program creates and maps a global page frame section.
7       *   Data in the section is accessed through an array.
8       *
9       ENVIRONMENT DIVISION.
10
11      DATA DIVISION.
12      WORKING-STORAGE SECTION.
13      01  SEC_FLAGS       PIC S9(9) COMP.
14      01  SEC$M_PAGFIL    PIC S9(9) COMP
15                           VALUE EXTERNAL SEC$M_PAGFIL.
16      01  SEC$M_GBL       PIC S9(9) COMP
17                           VALUE EXTERNAL SEC$M_GBL.
18      01  SEC$M_DZRO      PIC S9(9) COMP
19                           VALUE EXTERNAL SEC$M_DZRO.
20      01  SEC$M_WRT       PIC S9(9) COMP
21                           VALUE EXTERNAL SEC$M_WRT.
22      01  DATA_ARRAY EXTERNAL.
23          02  IARRAY      PIC S9(9) COMP  OCCURS 50 TIMES.
24      01  MY_ADR.
25          02  POINTER VALUE REFERENCE DATA_ARRAY.
26          02  POINTER VALUE REFERENCE MY_ADR.
27      01  SYS_ADR.
28          02  SYS_ADR_ELEM PIC S9(9) COMP  OCCURS  2 TIMES.
29      01  NAME            PIC  X(4)       VALUE 'GSEC'.
30      01  I               PIC S9(9) COMP.
31      01  OUT_IARRAY      PIC Z(9)9.
32      01  STAT            PIC S9(9) COMP.
33
34      PROCEDURE DIVISION.
35      BEGIN.
36      *
37      *   Associate with common cluster MYCLUS
38  (1)     CALL 'SYS$ASCEFC' USING BY VALUE 64
39                                  BY DESCRIPTOR 'MYCLUS'
40                                  BY VALUE 0 0
41                              GIVING STAT.
42          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
43      *
44      *   Create and map the temporary global section
45          COMPUTE SEC_FLAGS = SEC$M_PAGFIL+SEC$M_GBL+SEC$M_WRT+SEC$M_DZRO
46  (2)     CALL 'SYS$CRMPSC' USING BY REFERENCE MY_ADR SYS_ADR
47                                  BY VALUE 0 SEC_FLAGS
48                                  BY DESCRIPTOR NAME
49                                  BY VALUE 0 0 0 1 0 0 0
50                               GIVING STAT.
51          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
52      *
53      *   Manipulate the data in the global section
54          PERFORM VARYING I FROM 1 BY 1 UNTIL I > 50
55  (3)             MOVE I TO IARRAY(I)
56          END-PERFORM.
57      *
58      *   Wait for PAGEFIL2 to update the section
59          CALL 'SYS$SETEF' USING BY VALUE 72
60                           GIVING STAT.
61          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
62          DISPLAY 'Waiting for PAGEFIL2 to update section.'.
63          CALL 'SYS$WAITFR' USING BY VALUE 73
64                            GIVING STAT.
65          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
```

Example 2 Communicating With Global Pagefile Sections (Sheet 1 of 3)

```
66      *
67      *   Print the modified pages
68          DISPLAY 'Modified data in the global section:'
69          PERFORM VARYING I FROM 1 BY 1 UNTIL I > 50
70                  MOVE IARRAY(I) TO OUT_IARRAY
71                  DISPLAY OUT_IARRAY WITH NO ADVANCING
72          END-PERFORM
73          STOP RUN.


 1      *                                              PAGEFIL2.COB
 2      IDENTIFICATION DIVISION.
 3      *
 4      PROGRAM-ID. PAGEFIL2.
 5      *
 6      *   This program maps and modifies a global section
 7      *   after PAGEFIL1 creates the section.  Programs
 8      *   PAGEFIL1 and PAGEFIL2 synchronize the processing
 9      *   of the global page frame section through the use
10      *   of common event flags.
11      *
12      DATA DIVISION.
13      WORKING-STORAGE SECTION.
14
15      01  SEC$M_WRT        PIC S9(9) COMP
16                            VALUE EXTERNAL SEC$M_WRT.
17      01  DATA_ARRAY EXTERNAL.
18          02  IARRAY       PIC S9(9) COMP  OCCURS 50 TIMES.
19      01  MY_ADR.
20          02  POINTER VALUE REFERENCE DATA_ARRAY.
21          02  POINTER VALUE REFERENCE MY_ADR.
22      01  I                PIC S9(9) COMP.
23      01  OUT_IARRAY       PIC Z(9)9.
24      01  STAT             PIC S9(9) COMP.
25
26      PROCEDURE DIVISION.
27      BEGIN.
28      *
29      *   Associate with common cluster MYCLUS and wait for
30      *   event flag to be set
31  (1)     CALL 'SYS$ASCEFC' USING BY VALUE 64
32                                  BY DESCRIPTOR 'MYCLUS'
33                                  BY VALUE 0 0
34                              GIVING STAT.
35          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
36          CALL 'SYS$WAITFR' USING BY VALUE 72
37                            GIVING STAT.
38          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
39      *
40      *   Map the global section
41          CALL 'SYS$MGBLSC' USING BY REFERENCE MY_ADR
42  (2)                             BY VALUE 0 0 SEC$M_WRT
43                                  BY DESCRIPTOR 'GSEC'
44                                  BY VALUE 0 0
45                             GIVING STAT.
46          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
47      *
48      *   Print out the data in the global section and
49      *   multiply each value by two
```

Example 2 Communicating With Global Pagefile Sections (Sheet 2 of 3)

```
    50          DISPLAY 'Original data in the global section:'.
    51          PERFORM VARYING I FROM 1 BY 1 UNTIL I > 50
    52                  MOVE IARRAY(I) TO OUT_IARRAY
    53                  DISPLAY OUT_IARRAY WITH NO ADVANCING
    54                  MULTIPLY 2 BY IARRAY(I)
    55          END-PERFORM.
    56      *
    57      *   Set an event flag to allow PAGEFIL1 to continue execution
    58          CALL 'SYS$SETEF' USING BY VALUE 73
    59                           GIVING STAT.
    60          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
    61          STOP RUN.

     1  ❺  !                                              LINKOPT.OPT
     2      PSECT=DATA_ARRAY,PAGE


$
$ COBOL PAGEFIL1
$ LINK PAGEFIL1,LINKOPT/OPT
$
$ RUN PAGEFIL1
Waiting for PAGEFIL2 to update section.
Modified data in the global section:
         2         4         6         8        10        12        14        16
        18        20        22        24        26        28        30        32
        34        36        38        40        42        44        46        48
        50        52        54        56        58        60        62        64
        66        68        70        72        74        76        78        80
        82        84        86        88        90        92        94        96
        98       100


$ ! RUN FROM ANOTHER TERMINAL
$
$ COBOL PAGEFIL2
$ LINK PAGEFIL2,LINKOPT/OPT
$
$ RUN PAGEFIL2
Original data in the global section:
         1         2         3         4         5         6         7         8
         9        10        11        12        13        14        15        16
        17        18        19        20        21        22        23        24
        25        26        27        28        29        30        31        32
        33        34        35        36        37        38        39        40
        41        42        43        44        45        46        47        48
        49        50
$
```

Example 2 Communicating With Global Pagefile Sections (Sheet 3 of 3)

Example 3

Example 3 shows how to create a global section and use it to transfer data between two processes. In this example you must run programs GLOBAL1 and GLOBAL2 in the same UIC group. Since GLOBAL1 creates the global section, it must be run before GLOBAL2.

❶ A MACRO procedure, GET_CHAN, is used to create and open the file SECTION.DAT. A channel number must also be obtained. This procedure must be assembled and linked with the COBOL program GLOBAL1.

❷ To pass the channel number from GET_CHAN to GLOBAL1, GET_CHAN is called as a function. The function returns a value to the COBOL variable SEC_CHAN.

❸ Once the file has been opened and a channel number returned, the $CRMPSC system service creates and maps the global section.

The starting and ending virtual addresses of the section are placed in MY_ADR. The output argument SYS_ADR receives the starting and ending system virtual addresses. The flag SEC$M_GLOBAL requests a global section. The flag SEC$M_WRT indicates that the pages should be writable as well as readable. The SEC$M_DZRO flag requests pages filled with zeros.

❹ GLOBAL2 maps the existing section as writable by specifying the SEC$M_WRT flag. Note that the SEC$M_DZRO flag is not set, since that would destroy any data in the section.

❺ The $CRMPSC system service maps pages starting at page boundaries. Since data in COBOL is longword (not page) aligned, you must ensure that IARRAY starts on a page boundary. The PSECT construct in the options file for the LINKER accomplishes this. GLOBAL1 and GLOBAL2 are linked with the options file.

```
1       *                                                GLOBAL1.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. GLOBAL1.
5       *
6       *   This program creates and maps a global section. Data
7       *   in the section file is accessed through an array.
8       *
9       ENVIRONMENT DIVISION.
10      INPUT-OUTPUT SECTION.
11      FILE-CONTROL.
12
13          SELECT SECTION_FILE ASSIGN TO 'SECTION'.
14
15      DATA DIVISION.
16      FILE SECTION.
17      FD  SECTION_FILE.
18      01  SECTION_LINE    PIC X(80).
19
20      WORKING-STORAGE SECTION.
21      01  SEC_FLAGS       PIC S9(9) COMP.
22      01  SEC$M_GBL       PIC S9(9) COMP
23                          VALUE EXTERNAL SEC$M_GBL.
24      01  SEC$M_DZRO      PIC S9(9) COMP
25                          VALUE EXTERNAL SEC$M_DZRO.
26      01  SEC$M_WRT       PIC S9(9) COMP
27                          VALUE EXTERNAL SEC$M_WRT.
28      01  DATA_ARRAY EXTERNAL.
29          02 IARRAY       PIC S9(9) COMP  OCCURS 50 TIMES.
30      01  MY_ADR.
31          02 POINTER VALUE REFERENCE DATA_ARRAY.
32          02 POINTER VALUE REFERENCE MY_ADR.
33      01  SYS_ADR.
34          02 SYS_ADR_ELEM PIC S9(9) COMP  OCCURS  2 TIMES.
35      01  NAME            PIC  X(4)       VALUE 'GSEC'.
36      01  SEC_CHAN        PIC  9(9) COMP.
37      01  I               PIC S9(9) COMP.
38      01  OUT_IARRAY      PIC Z(9)9.
39      01  STAT            PIC S9(9) COMP.
40      01  IOSB.
41          02 IOSB1        PIC S9(9)  COMP VALUE 0.
42          02 IOSB2        PIC S9(9)  COMP VALUE 0.
43
44      PROCEDURE DIVISION.
45      BEGIN.
46      *
47      *   Associate with common cluster MYCLUS
48          CALL 'SYS$ASCEFC' USING BY VALUE 64
49                                  BY DESCRIPTOR 'MYCLUS'
50                                  BY VALUE 0 0
51                              GIVING STAT.
52          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
53      *
54      *   Call MACRO procedure to open the file and return channel
55  (2)     CALL 'GET_CHAN' GIVING SEC_CHAN.
56      *
57      *   Create and map the temporary global section
58          COMPUTE SEC_FLAGS = SEC$M_GBL + SEC$M_WRT + SEC$M_DZRO
59  (3)     CALL 'SYS$CRMPSC' USING BY REFERENCE MY_ADR SYS_ADR
60                                  BY VALUE 0 SEC_FLAGS
61                                  BY DESCRIPTOR NAME
62                                  BY VALUE 0 0 SEC_CHAN 1 0 0 0
63                              GIVING STAT.
64          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
```

Example 3 Communicating With Global Sections (Sheet 1 of 4)

```
65      *
66      *   Manipulate the data in the global section
67          PERFORM VARYING I FROM 1 BY 1 UNTIL I > 50
68                  MOVE I TO IARRAY(I)
69          END-PERFORM.
70      *
71      *   Wait for GLOBAL2 to update the section
72          CALL 'SYS$SETEF' USING BY VALUE 72
73                           GIVING STAT.
74          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
75          DISPLAY 'Waiting for GLOBAL2 to update section.'.
76          CALL 'SYS$WAITFR' USING BY VALUE 73
77                           GIVING STAT.
78          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
79      *
80      *   Print the modified pages
81          DISPLAY 'Modified data in the global section:'
82          PERFORM VARYING I FROM 1 BY 1 UNTIL I > 50
83                  MOVE IARRAY(I) TO OUT_IARRAY
84                  DISPLAY OUT_IARRAY WITH NO ADVANCING
85          END-PERFORM
86          STOP RUN.


 1      *                                                GLOBAL2.COB
 2      IDENTIFICATION DIVISION.
 3      *
 4      PROGRAM-ID. GLOBAL2.
 5      *
 6      *   This program maps and modifies a global section
 7      *   after GLOBAL1 creates the section.  Programs
 8      *   GLOBAL1 and GLOBAL2 synchronize the processing
 9      *   of the global section through the use of common
10      *   event flags.
11      *
12      DATA DIVISION.
13      WORKING-STORAGE SECTION.
14
15      01  SEC$M_WRT          PIC S9(9) COMP
16                              VALUE EXTERNAL SEC$M_WRT.
17      01  DATA_ARRAY EXTERNAL.
18          02 IARRAY          PIC S9(9) COMP  OCCURS 50 TIMES.
19      01  MY_ADR.
20          02 POINTER VALUE REFERENCE DATA_ARRAY.
21          02 POINTER VALUE REFERENCE MY_ADR.
22      01  I                  PIC S9(9) COMP.
23      01  OUT_IARRAY         PIC Z(9)9.
24      01  STAT               PIC S9(9) COMP.
25
```

Example 3 Communicating With Global Sections (Sheet 2 of 4)

```
26      PROCEDURE DIVISION.
27      BEGIN.
28      *
29      *   Associate with common cluster MYCLUS and wait for
30      *   event flag to be set
31          CALL 'SYS$ASCEFC' USING BY VALUE 64
32                                  BY DESCRIPTOR 'MYCLUS'
33                                  BY VALUE 0 0
34                           GIVING STAT.
35          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
36          CALL 'SYS$WAITFR' USING BY VALUE 72
37                           GIVING STAT.
38          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
39      *
40      *   Map the global section
41          CALL 'SYS$MGBLSC' USING BY REFERENCE MY_ADR
42      (4)                         BY VALUE 0 0 SEC$M_WRT
43                                  BY DESCRIPTOR 'GSEC'
44                                  BY VALUE 0 0
45                           GIVING STAT.
46          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
47      *
48      *   Print out the data in the global section and
49      *   multiply each value by two
50          DISPLAY 'Original data in the global section:'.
51          PERFORM VARYING I FROM 1 BY 1 UNTIL I > 50
52                  MOVE IARRAY(I) TO OUT_IARRAY
53                  DISPLAY OUT_IARRAY WITH NO ADVANCING
54                  MULTIPLY 2 BY IARRAY(I)
55          END-PERFORM.
56      *
57      *   Set an event flag to allow GLOBAL1 to continue execution
58          CALL 'SYS$SETEF' USING BY VALUE 73
59                           GIVING STAT.
60          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
61          STOP RUN.
```

```
 1  (1)  ;                                              GETCHAN.MAR
 2       ;   This MACRO routine OPENs a file using RMS macros,
 3       ;   and returns a channel number to the file in location
 4       ;   SEC_CHAN.
 5       ;
 6
 7       MY_FAB:     $FAB    FNM=<SECTION.DAT>, ALQ=<1>, FOP=<CTG,UFO>
 8       MY_RAB:     $RAB    FAB=MY_FAB
 9
10           .PSECT CHANNEL,PIC,OVR,GBL,SHR,NOEXE,REL,WRT,LONG
11       SEC_CHAN:
12           .BLKL   1
13       ;
14       ;   This PSECT contains the code to open the file
15           .PSECT
16           .ENTRY  GET_CHAN,^M<R2>         ;R2 will be destroyed
17       ;
18       ;   Open file
19           $CREATE         FAB=MY_FAB
20       ;
21       ;   Get address of FAB
22           MOVAL           MY_FAB,R2
23       ;
24       ;   Return channel to the file
25           MOVZWL          FAB$L_STV(R2),SEC_CHAN
26           MOVL            SEC_CHAN,R0
27           RET
28           .END
```

Example 3 Communicating With Global Sections (Sheet 3 of 4)

```
    1   ❺  !                                          LINKOPT.OPT
    2      PSECT=DATA_ARRAY,PAGE


$ ! RUN FROM ONE TERMINAL
$
$ COBOL GLOBAL1
$ MACRO GETCHAN
$ LINK GLOBAL1,GETCHAN,LINKOPT/OPT
$
$ RUN GLOBAL1
Waiting for GLOBAL2 to update section.
Modified data in the global section:
         2          4          6          8         10         12         14         16
        18         20         22         24         26         28         30         32
        34         36         38         40         42         44         46         48
        50         52         54         56         58         60         62         64
        66         68         70         72         74         76         78         80
        82         84         86         88         90         92         94         96
        98        100


$ ! RUN FROM ANOTHER TERMINAL
$
$ COBOL GLOBAL2
$ LINK GLOBAL2,LINKOPT/OPT
$
$ RUN GLOBAL2
Original data in the global section:
         1          2          3          4          5          6          7          8
         9         10         11         12         13         14         15         16
        17         18         19         20         21         22         23         24
        25         26         27         28         29         30         31         32
        33         34         35         36         37         38         39         40
        41         42         43         44         45         46         47         48
        49         50
$
```

Example 3 Communicating With Global Sections (Sheet 4 of 4)

Example 4

Task-to-task communication consists of four steps.

1. The source task establishes a logical link.

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT file-name ASSIGN TO TARGET_TASK_NAME.
                    .
                    .
                    .
PROCEDURE DIVISION.
BEGIN.
     OPEN OUTPUT file-name.
```

2. The target task completes the logical link.

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT file-name ASSIGN TO SYS$NET.
                    .
                    .
                    .
PROCEDURE DIVISION.
BEGIN.
     OPEN INPUT file-name.
```

3. Data is exchanged using READ/WRITE statements.
4. Either task can disconnect the logical link.

```
CLOSE file-name.
```

Example 5

Example 5 shows how one process can use task-to-task communication to run a process on another processor and communicate with it, providing that the two processors are linked by transparent DECnet.

❶ For this example you must have an account on another processor (node 2) linked by DECnet to your local node (node 1). Before you run TASK1, create the following files in your login default directory on node 2:

TASK2.EXE executable image of TASK2

TASK2.COM a command procedure file, containing the following two command lines:

```
$ RUN TASK2
$ PURGE TASK2.LOG
```

Make the following logical name assignment on node 1:

```
$ ASSIGN node"""username password""":::"""TASK=TASK2""" REMOTE
```

This DCL command assigns the logical name REMOTE to TASK2 (the program to be run on the remote node). Note that the correct username and password must be specified.

To test the example, run TASK 1 from your account on node 1. For test purposes, node 1 and node 2 can be the same processor.

❷ TASK1 uses the COBOL OPEN statement to request a logical link connection to TASK2. The filename parameter specifies the logical name REMOTE, which has been assigned to the remote task, TASK2.

❸ TASK2.COM is invoked on the remote node. It uses the DCL RUN command to run TASK2.EXE.

❹ TASK2 completes the logical link via SYS$NET.

❺ Data is sent by TASK1 and and received by TASK2.

❻ The logical link is disconnected by either TASK1 or TASK2.

Debugging Hint

The following steps are useful for debugging a program on a remote node:

1. Compile and link TASK2 with the debugger:

```
$ COBOL/DEBUG TASK2
$ LINK/DEBUG TASK2
```

2. In TASK2.COM, assign an unused terminal on the remote node to DBG$INPUT and DBG$OUTPUT:

```
$ ASSIGN TTxx DBG$INPUT
$ ASSIGN TTxx DBG$OUTPUT
```

   The debugger will appear on the terminal you specified so that TASK2 can be debugged.

```
 1        *                                                TASK1.COB
 2        IDENTIFICATION DIVISION.
 3        *
 4        PROGRAM-ID. TASK1.
 5        *
 6        *   This program illustrates task to task
 7        *   communication.  Prior to running this program
 8        *   the logical name REMOTE must be defined:
 9        *
10    (1) * $ ASSIGN node"""username password""":::""TASK=TASK2""" -
11        * $_REMOTE
12        *
13        ENVIRONMENT DIVISION.
14        INPUT-OUTPUT SECTION.
15        FILE-CONTROL.
16
17            SELECT REMOTE_NODE_OUTPUT ASSIGN TO REMOTE.
18
19        DATA DIVISION.
20        FILE SECTION.
21        FD  REMOTE_NODE_OUTPUT.
22        01  O                PIC S9(9) COMP.
23
24        WORKING-STORAGE SECTION.
25        01  J                PIC S9(9) COMP.
26        01  OUT-J            PIC ZZ9.
27
28        PROCEDURE DIVISION.
29        BEGIN.
30        *
31        *   Establish network link
32    (2)     OPEN OUTPUT REMOTE_NODE_OUTPUT
33            PERFORM VARYING J FROM 1 BY 1 UNTIL J > 10
34        *
35        *           Send a message to other task
36                  (WRITE O FROM J
37    (5)           {MOVE J TO OUT_J
38                  (DISPLAY 'Sent message ', OUT_J
39            END-PERFORM
40        *
41        *   Disconnect the network link
42    (6)     CLOSE REMOTE_NODE_OUTPUT.
43            STOP RUN.


 1        *                                                TASK2.COB
 2        IDENTIFICATION DIVISION.
 3        *
 4        PROGRAM-ID. TASK2.
 5        *
 6        *   This program illustrates task to task
 7        *   communication. It is run by TASK1 via
 8        *   the command procedure TASK2.COM.
 9        *
10        ENVIRONMENT DIVISION.
11        INPUT-OUTPUT SECTION.
12        FILE-CONTROL.
13
14            SELECT NETWORK_NODE_INPUT ASSIGN TO SYS$NET.
15
16        DATA DIVISION.
17        FILE SECTION.
18        FD  NETWORK_NODE_INPUT.
19        01  J                PIC S9(9) COMP.
20
21        WORKING-STORAGE SECTION.
22        01  OUT_J            PIC ZZ9.
23
```

Example 5 Task-to-Task Communication (Sheet 1 of 2)

```
 24        PROCEDURE DIVISION.
 25        BEGIN.
 26        *
 27        *    Connect to network link
 28     4      OPEN INPUT NETWORK_NODE_INPUT.
 29            PERFORM 10 TIMES
 30        *
 31        *             Read message from parent task
 32                    READ NETWORK_NODE_INPUT AT END CONTINUE
 33     5              END-READ
 34                    MOVE J TO OUT_J
 35                    DISPLAY 'Received value' OUT_J
 36            END-PERFORM
 37        *
 38        *    Disconnect the network link
 39     6      CLOSE NETWORK_NODE_INPUT.
 40            STOP RUN.


  1     3   $!                                                TASK2.COM
  2         $!
  3         $!  This command procedure is invoked by TASK1 to
  4         $!  establish task to task communication between
  5         $!  itself and TASK2.
  6         $!
  7         $ RUN TASK2
  8         $ PURGE TASK2.LOG

$ COBOL TASK1
$ LINK TASK1
$
$ ! Assign the logical name REMOTE to the target task, TASK2.
$ ASSIGN SUPER"""EDSERVICES COURSE""":::"""TASK=TASK2""" REMOTE
$ RUN TASK1
Sent message   1
Sent message   2
Sent message   3
Sent message   4
Sent message   5
Sent message   6
Sent message   7
Sent message   8
Sent message   9
Sent message  10
$
```

Example 5 Task-to-Task Communication (Sheet 2 of 2)

Lab Exercises

1. Refer to the example programs MAILS.COB and MAILR.COB in this module to complete the exercise.

   a. Modify the MAILS example to complete the mailbox write immediately. (Hint: Use the function modifier IO$M_NOW — refer to the chapter on the Mailbox Driver in the *VAX/VMS I/O Users Reference Manual-Part 1)*

   b. Modify the MAILR example to output the PID of the process that sent the message, in addition to outputting the mailbox message. (Hint: Declare an I/O status block for the SYS$QIOW. Output the PID in hexidecimal.)

2. The system services and Run-Time Library routines listed below are used to create, translate, and delete supervisor mode logical names, and to write to and read from P1 Common.

```
$CRELNM
$TRNLNM
$DELLNM

LIB$PUT_COMMON
LIB$GET_COMMON
```

Use these routines to write two programs as follows:

a. A program (LABSOL2A.COB) that defines a supervisor mode logical name BKP: to be the equivalence string MTA0:, and then writes a short message to P1 Common .

b. A program (LABSOL2B.COB) that accesses the logical name BKP: and reads the message in P1 Common. The program should print the translation for BKP:, the length of the equivalence string, the name of the logical name table used, and the access mode. It should also display the message from P1 Common and its length.

3. Study Example 3 (Global sections) in this module, then write a program that creates and uses a private section. The program should create a file then map it as a private section. The section characteristics should be read/write and demand-zero pages. Once the section has been created, write the ASCII codes for the digits 0 through 9 (hex 30 through 39) in the first ten bytes of the private section. Print out what was written.

   Your program should:

   a. Use the USEROPEN keyword speciying the entry address of a useropen routine. This routine should return the file's channel number.

   b. Define a common area MYSECT to be mapped, and link your program with an options file.

Lab Solutions

1.

a.

```
1       *                                                LABSOL1A.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID.    LABSOL1A.
5       *
6       *   This program will write to a mailbox with the
7       *   logical name FOZZIE_BEAR and complete the operation
8       *   immediately. Thus, the write will complete before
9       *   the process LABSOL1B reads the mailbox.
10      *
11      ENVIRONMENT DIVISION.
12      INPUT-OUTPUT SECTION.
13      FILE-CONTROL.
14                  SELECT MAILBOX_NAME ASSIGN TO 'FOZZIE_BEAR'.
15
16      DATA DIVISION.
17      FILE SECTION.
18
19      FD  MAILBOX_NAME.
20      01  MESAGE          PIC X(18).
21
22      WORKING-STORAGE SECTION.
23
24      01  CHANNEL         PIC 9(4)  COMP  VALUE  0.
25      01  MESSAGE_TEXT    PIC X(18) VALUE 'This is a message.'.
26      01  FUNC_CODE       PIC S9(9) COMP  VALUE  0.
27      01  IO$_WRITEVBLK   PIC S9(9) COMP  VALUE EXTERNAL IO$_WRITEVBLK.
28      01  IO$M_NOW        PIC S9(9) COMP VALUE EXTERNAL IO$M_NOW.
29      01  STAT            PIC S9(9) COMP  VALUE 0.
30      01  IOSB.
31          02  IOSB1       PIC S9(9)  COMP  VALUE 0.
32          02  IOSB2       PIC S9(9)  COMP  VALUE 0.
33
34      PROCEDURE DIVISION.
35      BEGIN.
36      *
37      *   Create mailbox and/or assign a channel to it
38          CALL 'SYS$CREMBX' USING BY VALUE 0,
39                                  BY REFERENCE CHANNEL
40                                  BY VALUE 0 0 0 0
41                                  BY DESCRIPTOR 'FOZZIE_BEAR',
42                              GIVING STAT.
43          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
44      *
45      *   Write a message to the mailbox
46          ADD IO$_WRITEVBLK IO$M_NOW GIVING FUNC_CODE
47          MOVE MESSAGE_TEXT TO MESAGE.
48          CALL 'SYS$QIOW' USING BY VALUE 2 CHANNEL FUNC_CODE
49                                BY REFERENCE IOSB
50                                BY VALUE 0 0
51                                BY REFERENCE MESAGE
52                                BY VALUE 18 0 0 0 0
53                            GIVING STAT.
54          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
55          IF IOSB1 IS FAILURE CALL 'LIB$STOP' USING BY VALUE IOSB1.
56      *
57      *   Notify user
58          DISPLAY 'LABSOL1A has completed mailbox write.'.
59          STOP RUN.
```

b.

```
1       *                                                    LABSOL1B.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID.    LABSOL1B.
5       *
6       *   This program will read the mailbox created
7       *   by LABSOL1A.
8       *
9       ENVIRONMENT DIVISION.
10      INPUT-OUTPUT SECTION.
11      FILE-CONTROL.
12                  SELECT MAILBOX ASSIGN TO 'FOZZIE_BEAR'.
13
14      DATA DIVISION.
15      FILE SECTION.
16
17      FD  MAILBOX.
18      01  MESAGE          PIC X(18).
19      WORKING-STORAGE SECTION.
20
21      01  CHANNEL         PIC 9(4)  COMP  VALUE  0.
22      01  MAILBOX_NAME    PIC X(11)       VALUE 'FOZZIE_BEAR'.
23      01  IO$_READVBLK    PIC S9(9) COMP  VALUE EXTERNAL IO$_READVBLK.
24      01  STAT            PIC S9(9) COMP  VALUE 0.
25      01  MAILBOX_IOSB.
26          02  M_IOSB1     PIC S9(9) COMP  VALUE 0.
27          02  M_IOSB2     PIC S9(9) COMP  VALUE 0.
28      01  PID             PIC 9(9).
29      01 INPUTX           PIC X(30)       VALUE SPACES.
30
31      PROCEDURE DIVISION.
32      BEGIN.
33      *
34      *   Create and/or assign a channel to a mailbox
35          CALL 'SYS$CREMBX' USING BY VALUE 0
36                                  BY REFERENCE CHANNEL
37                                  BY VALUE 0 0 0 0
38                                  BY DESCRIPTOR MAILBOX_NAME
39                            GIVING STAT.
40          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
41      *
42      *   Read the mailbox
43          CALL 'SYS$QIOW' USING BY VALUE 2 CHANNEL IO$_READVBLK
44                                BY REFERENCE MAILBOX_IOSB
45                                BY VALUE 0 0
46                                BY REFERENCE INPUTX
47                                BY VALUE 30 0 0 0 0
48                          GIVING STAT.
49          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
50          IF M_IOSB1 IS FAILURE CALL 'LIB$STOP' USING BY VALUE M_IOSB1.
51      *
52      *   Convert PID to hexidecimal
53          CALL  'OTS$CVT_L_TZ' USING BY REFERENCE M_IOSB2
54                                     BY DESCRIPTOR PID
55                                     BY VALUE 8 4
56                               GIVING STAT.
57          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
58      *   Output the mailbox message
59          DISPLAY INPUTX.
60          DISPLAY 'The message was sent by  ' PID.
61          STOP RUN.
```

2.

a.

```
1       *                                                LABSOL2A.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID.    LABSOL2A.
5       *
6       *   This program will write create a supervisor mode
7       *   logical name and write a message to P1 common.
8       *
9       DATA DIVISION.
10      WORKING-STORAGE SECTION.
11
12      01  LOGICAL_NAME  PIC X(4)  VALUE 'BKP:'.
13      01  EQUIV_NAME    PIC X(5)  VALUE 'MTA0:'.
14      01  EQUIV_LEN         PIC 9(4)      COMP VALUE 0.
15      01  TABLE_NAME        PIC X(16)     VALUE 'LNM$SYSTEM_TABLE'.
16      01  COM_MESSAGE       PIC X(18)     VALUE 'This is a message.'.
17      01  STAT          PIC S9(9)         COMP VALUE 0.
18      01  CRELNM_LIST.
19          02  LNM_STRING.
20          03      PIC S9(4)       COMP VALUE 4.
21          03      PIC S9(4)       COMP VALUE 2.
22          03      POINTER VALUE REFERENCE EQUIV_NAME.
23          03      POINTER VALUE REFERENCE EQUIV_LEN.
24      PROCEDURE DIVISION.
25      BEGIN.
26      *
27      *   Define supervisor mode logical name
28          CALL 'SYS$CRELNM' USING BY VALUE 0
29                                  BY DESCRIPTOR TABLE_NAME LOGICAL_NAME
30                                  BY VALUE 0
31                                  BY REFERENCE CRELNM_LIST
32                                    GIVING STAT.
33          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
34      *
35      *   Write a message to P1 Common
36          CALL 'LIB$PUT_COMMON' USING BY DESCRIPTOR COM_MESSAGE
37                                      BY VALUE 0
38                                  GIVING STAT.
39          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
40      *
41          STOP RUN.
```

b.

```
1       *                                                    LABSOL2B.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID.    LABSOL2B.
5       *
6       *   This program will access, display, and
7       *   delete a supervisor mode logical name and
8       *   read a message from P1 common.
9       *
10      DATA DIVISION.
11      WORKING-STORAGE SECTION.
12      01  PSL$C_SUPER   PIC S9(9) COMP VALUE 2.
13      01  LOGICAL_NAME  PIC X(4)  VALUE 'BKP:'.
14      01  TABLE_NAME        PIC X(16)     VALUE 'LNM$SYSTEM_TABLE'.
15      01  STRING_LEN    PIC S9(4) COMP VALUE 255.
16      01  STRING_DEST   PIC X(255).
17      01  RET_LENGTH        PIC S9(9)     COMP.
18      01  RET_LENGTH_LEN          PIC S9(4) COMP.
19      01  RET_TABLE         PIC X(255).
20      01  RET_TABLE_LEN           PIC S9(4) COMP.
21      01  ACCESS_MODE   PIC 9(9)  COMP.
22      01  COM_NUM       PIC 9(9)  COMP.
23      01  DISP_NUM      PIC Z(9).
24      01  COM_MESSAGE       PIC X(252).
25      01  STAT          PIC S9(9)         COMP VALUE 0.
26      01  TRNLNM_LIST.
27          02  LNM_STRING.
28          03      PIC S9(4)       COMP VALUE 255.
29      *   03      PIC S9(4)       COMP VALUE LNM$_STRING.
30          03      PIC S9(4)       COMP VALUE 2.
31          03      POINTER VALUE REFERENCE STRING_DEST.
32          03      POINTER VALUE REFERENCE STRING_LEN.
33          02  LNM_LENGTH.
34          03      PIC S9(4)       COMP VALUE 4.
35      *   03      PIC S9(4)       COMP VALUE EXTERNAL LNM$_LENGTH.
36          03      PIC S9(4)       COMP VALUE 5.
37          03      POINTER VALUE REFERENCE RET_LENGTH.
38          03      POINTER VALUE REFERENCE RET_LENGTH_LEN.
39          02  LNM_TABLE.
40          03      PIC S9(4)       COMP VALUE 255.
41      *   03      PIC S9(4)       COMP VALUE EXTERNAL LNM$_TABLE.
42          03      PIC S9(4)       COMP VALUE 4.
43          03      POINTER VALUE REFERENCE RET_TABLE.
44          03      POINTER VALUE REFERENCE RET_TABLE_LEN.
45
```

(Sheet 1 of 2)

```
46        PROCEDURE DIVISION.
47        BEGIN.
48        *
49        *   Access contents of supervisor mode logical name
50            CALL 'SYS$TRNLNM' USING BY VALUE 0
51                                    BY DESCRIPTOR TABLE_NAME LOGICAL_NAME
52                                    BY VALUE 0
53                                    BY REFERENCE TRNLNM_LIST
54                                      GIVING STAT.
55            IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
56        *
57        *   Display equivalence string and other parameters
58            DISPLAY 'BKP: ==>          ' STRING_DEST
59            MOVE RET_LENGTH_LEN TO DISP_NUM
60            DISPLAY 'Equivalence string length is    ' DISP_NUM
61            DISPLAY 'Logical name table is ', RET_TABLE
62        *
63        *   Delete logical name
64            CALL 'SYS$DELLNM' USING BY DESCRIPTOR TABLE_NAME LOGICAL_NAME
65                                    BY REFERENCE PSL$C_SUPER
66                          GIVING STAT.
67            IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
68        *
69        *   Read message from P1 Common
70            CALL 'LIB$GET_COMMON' USING BY DESCRIPTOR COM_MESSAGE
71                                        BY REFERENCE COM_NUM
72                                    GIVING STAT.
73            IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
74        *
75        *   Display message
76            DISPLAY SPACES
77            DISPLAY COM_MESSAGE
78            MOVE COM_NUM TO DISP_NUM.
79            DISPLAY 'Message length = ' DISP_NUM.
80        *
81            STOP RUN.
```

(Sheet 2 of 2)

3.

```
1       *                                                    LABSOL3.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL3.
5       *
6       *
7       DATA DIVISION.
8       WORKING-STORAGE SECTION.
9
10      01  SEC_FLAGS        PIC S9(9) COMP.
11      01  SEC$M_DZRO       PIC S9(9) COMP  VALUE EXTERNAL SEC$M_DZRO.
12      01  SEC$M_WRT        PIC S9(9) COMP  VALUE EXTERNAL SEC$M_WRT.
13      01  DATA_ARRAY EXTERNAL.
14          02 IARRAY        PIC 9   OCCURS 10 TIMES.
15          02 END_OF_ARRAY PIC X.
16      01  MAPRANGE.
17         02       POINTER VALUE REFERENCE DATA_ARRAY.
18         02       POINTER VALUE REFERENCE END_OF_ARRAY.
19      01  RETRANGE.
20         02       RET_ADDRESS     PIC S9(9) COMP  OCCURS  2 TIMES.
21      01  NAME             PIC  X(4)       VALUE 'GSEC'.
22      01  I                PIC  99         COMP.
23      01  ARRAY_DISP       PIC  9.
24      01  SEC_CHAN         PIC  9(9) COMP.
25      01  STAT             PIC S9(9) COMP.
26
27      PROCEDURE DIVISION.
28      BEGIN.
29      *
30      *   Call MACRO procedure to open the file and return the channel
31          CALL 'LAB3' GIVING SEC_CHAN.
32      *
33      *   Create and map the temporary global section
34          COMPUTE SEC_FLAGS = SEC$M_WRT + SEC$M_DZRO
35          CALL 'SYS$CRMPSC' USING BY REFERENCE MAPRANGE RETRANGE
36                                  BY VALUE 0 SEC_FLAGS
37                                  BY DESCRIPTOR NAME
38                                  BY VALUE 0 0 SEC_CHAN 1 0 0 0
39                           GIVING STAT.
40          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
41      *
42      *   Write to section
43          PERFORM VARYING I FROM 0 BY 1 UNTIL I > 9
44                  MOVE I TO IARRAY(I + 1)
45                  MOVE IARRAY(I + 1) TO ARRAY_DISP
46                  DISPLAY ARRAY_DISP WITH NO ADVANCING
47          END-PERFORM.
48          STOP RUN.
49      END PROGRAM LABSOL3.
```

Creating and Managing Other Processes

File Locations

On-line files are provided to help you master this module. The files are located in the directory:

```
DISK$COURSE:[COURSE.V4PROG.COB.MNAG]
```

For your convenience, your system manager may have created the following logical name equivalence:

```
DISK$COURSE:[COURSE.V4PROG.COB.MNAG] = V4PROG$COB$MNAG
```

Two types of files are provided:

1. The Module Programming Examples, which are also listed in this language workbook.
2. Solutions to the Laboratory Exercises.

If you wish to modify and/or compile, link, and run any of the programs, first copy the file into your directory.

Example 1

Example 1 illustrates the use of the $HIBER, $SCHDWK, and $CANWAK system services.

❶ A scheduled wake-up request is placed into the timer queue. When the $SCHDWK is issued, the process is not hibernating.

❷ The $HIBER system service forces the process into a hibernating state. At the end of five seconds, the scheduled wakeup will cause the process to be awakened.

❸ After performing a $HIBER three times, the scheduled wakeup is cancelled by the $CANWAK.

```
1       *                                                HIBER.COB
2       *
3       IDENTIFICATION DIVISION.
4       *
5       PROGRAM-ID. HIBER.
6       *
7       *   This program schedules itself to run in 5 seconds
8       *   and every 5 seconds thereafter.  The process then
9       *   goes into hibernation.  At the end of the third
10      *   scheduling, the wakeup is cancelled with $CANWAK.
11      *
12      DATA DIVISION.
13      WORKING-STORAGE SECTION.
14
15      01  STAT            PIC S9(9) COMP.
16      01  CURR-TIME       PIC X(23).
17      01  ALARM           PIC X(6)  VALUE '0 ::05'.
18      01  TIMER           PIC 9(18) COMP.
19
20      PROCEDURE DIVISION.
21      BEGIN.
22      *
23      *   Convert ALARM to system binary time
24          CALL 'SYS$BINTIM' USING BY DESCRIPTOR ALARM
25                                  BY REFERENCE TIMER
26                            GIVING STAT.
27          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
28      *
29      *   Schedule a wakeup request
30  ❶       CALL 'SYS$SCHDWK' USING BY VALUE 0 0
31                                  BY REFERENCE TIMER TIMER
32                            GIVING STAT.
33          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
34      *
35      *   Enter hibernate state three times
36          PERFORM 3 TIMES
37  ❷               CALL 'SYS$HIBER' GIVING STAT
38                  IF STAT IS FAILURE
39                          CALL 'LIB$STOP' USING BY VALUE STAT
40                  END-IF
```

Example 1 Scheduling and Canceling Wake-up Requests (Sheet 1 of 2)

```
41        *
42        *           Obtain the current time and display it
43                    CALL 'SYS$ASCTIM' USING BY VALUE 0
44                                            BY DESCRIPTOR CURR-TIME
45                                            BY VALUE 0 0
46                                       GIVING STAT
47                    IF STAT IS FAILURE
48                            CALL 'LIB$STOP' USING BY VALUE STAT
49                    END-IF
50                    DISPLAY CURR-TIME
51                END-PERFORM.
52
53        *
54        *   Cancel the scheduled wakeup request
55        3       CALL 'SYS$CANWAK' USING BY VALUE 0 0
56                                  GIVING STAT.
57                IF STAT IS FAILURE  CALL 'LIB$STOP' USING BY VALUE STAT.
58                DISPLAY 'Scheduled wakeup cancelled.'.
59                STOP RUN.
60
```

```
$ COBOL HIBER
$ LINK HIBER
$ RUN HIBER
16-DEC-1983 09:16:47.94
16-DEC-1983 09:16:52.94
16-DEC-1983 09:16:57.94
Scheduled wakeup cancelled.
$
```

Example 1 Scheduling and Canceling Wake-up Requests (Sheet 2 of 2)

Example 2

In Example 2, two processes communicate by means of a common event flag cluster, BERT. Only one process is interactive. It creates the other, a detached process named BIG_BIRD.

❶ To create a detached process you must specify a nonzero UIC argument to the $CREPRC system service.

For two processes to communicate by means of common event flags, they must belong to the same group. Therefore, before you run this program, change the UIC from [123,321] to your own UIC. (The group and member numbers 83 and 209 are decimal values for the octal numbers 123 and 321, respectively). Note that the first elementary item under UIC_MEMBER_GROUP is the member number, and the second elementary item is the group number. This is the order in which the group and member numbers are stored internally.

Note that you need DETACH privilege to create a detached process with a UIC different from your own. If you do not know your UIC, use the $SHOW PROCESS command to display your UIC. If your UIC is in the form of a character string, such as [VMSDEV,CLARK], use the lexical function F$IDENTIFIER to translate the user portion of your UIC from a character string to a number. Then use F$FAO to display the number in UIC format. For example:

```
$ X = F$IDENTIFIER ("CLARK", "NAME_TO_NUMBER")
$ SHOW SYMBOL X
   X=589840  HEX=00090010  OCTAL=00002200020
$ Y = F$FAO ("!%U", X)
$ SHOW SYMBOL Y
   Y = "[11,20]"
```

Refer to the *VAX/VMS DLC Dictionary* for more information about the lexical functions.

❷ The longword UIC must be passed by value. Two words (each specifying the decimal equivalent of an octal number) are initialized as a longword.

❸ If no value is specified for the BASPRI argument, the base priority of the created process defaults to 0 for a COBOL program. On a heavily loaded system, a process with a base priority of 0 might wait for a long time to be scheduled.

Typically, an interactive process has a base priority of 4. You can speed the scheduling of BIG_BIRD by using a value of 4 for the BASPRI argument of $CREPRC (as shown).

```
1       *                                                PROC1.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. PROC1.
5       *
6       *   This program illustrates detached process
7       *   creation.  It associates to common cluster BERT,
8       *   creates a detached process, then waits for the
9       *   second flag in the cluster to be set by the
10      *   created detached process.
11      *
12      DATA DIVISION.
13      WORKING-STORAGE SECTION.
14
15      01  STAT                    PIC S9(9) COMP.
16      01  CLUSTER_NAME            PIC X(4)  VALUE 'BERT'.
17      01  FILE_NAME               PIC X(5)  VALUE 'PROC2'.
18      01  PROCESS_NAME            PIC X(8)  VALUE 'BIG_BIRD'.
19  (1) 01  UIC_MEMBER_GROUP.
20  {       {    05  MEMBER-NUM     PIC 9(4)  COMP VALUE 209.
21  (2) {       05  GROUP-NUM      PIC 9(4)  COMP VALUE 83.
22      {01  UIC  REDEFINES UIC_MEMBER_GROUP PIC 9(9)   COMP.
23
24      PROCEDURE DIVISION.
25      BEGIN.
26      *
27      *   Associate cluster #2 as BERT
28          CALL 'SYS$ASCEFC' USING BY VALUE 64
29                                  BY DESCRIPTOR CLUSTER_NAME
30                                  BY VALUE 0 0
31                           GIVING STAT.
32          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
33      *
34      *   Clear the event flag to be set by PROC2
35          CALL 'SYS$CLREF' USING BY VALUE 65
36                           GIVING STAT.
37          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
38      *
39      *   Create the detached process
40          CALL 'SYS$CREPRC' USING BY VALUE 0
41                                  BY DESCRIPTOR FILE_NAME
42                                  BY VALUE 0 0 0 0 0
43                                  BY DESCRIPTOR PROCESS_NAME
44  (3)                             BY VALUE 4 UIC 0 0
45                           GIVING STAT.
46          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
47      *
48      *   Wait for the second flag to be set
49          DISPLAY 'Waiting for EFN 65 to be set.'.
50          CALL 'SYS$WAITFR' USING BY VALUE 65
51                           GIVING STAT.
52          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
53          DISPLAY 'EFN 65 has been set, exiting.'.
54          STOP RUN.
```

Example 2 Creating a Detached Process (Sheet 1 of 2)

```
 1      *                                                PROC2.COB
 2       IDENTIFICATION DIVISION.
 3       *
 4       PROGRAM-ID. PROC2.
 5       *
 6       *   This program will associate to common cluster
 7       *   BERT, then set the second flag in the cluster
 8       *   which will allow PROC1 to continue.
 9       *
10       DATA DIVISION.
11       WORKING-STORAGE SECTION.
12
13       01  STAT            PIC S9(9) COMP.
14       01  CLUSTER_NAME    PIC X(4)  VALUE 'BERT'.
15
16       PROCEDURE DIVISION.
17       BEGIN.
18       *
19       *   Associate cluster #3 as Bert
20           CALL 'SYS$ASCEFC' USING BY VALUE 96
21                                   BY DESCRIPTOR CLUSTER_NAME
22                                   BY VALUE 0 0
23                            GIVING STAT.
24           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
25       *
26       *   Set the second flag in BERT
27           CALL 'SYS$SETEF'  USING BY VALUE 97
28                            GIVING STAT.
29           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
30           STOP RUN.
```

```
$ COBOL PROC1, PROC2
$ LINK PROC1
$ LINK PROC2
$ SET PROCESS/PRIVILEGE=DETACH
$ RUN PROC1
Waiting for EFN 65 to be set.
EFN 65 has been set, exiting.
$
```

Example 2 Creating a Detached Process (Sheet 2 of 2)

Example 3

Example 3 illustrates the creation of a subprocess and the use of the $SETPRN system service. The $SETPRN system service allows the subprocess to know the name of the creating process. The creating process also defines SYS$OUTPUT for the subprocess, so they can share the same terminal.

❶ The program MAINPROC must translate the string TERMINAL, initially SYS$OUTPUT, to its lowest equivalence name form before passing it as an argument to $CREPRC.

❷ SYS$OUTPUT is a process permanent file. The VMS operating system identifies process permanent files by placing the ASCII characters <ESC><NUL> at the beginning of the equivalence string of the logical name. Bytes 3 and 4 of the equivalence string contain additional information for use by the system.

❸ $SETPRN gives the creating process a name that will be used by the subprocess to wake it.

VMS does not allow duplicate process names. Therefore, when running this program in lab, change the value of MAIN__NAME from MAINPROC to a unique process name. This will avoid any conflict caused by the situation where two students' programs simultaneously attempt to assign the same name to their own processes.

❹ When you run the image MAINPROC, your interactive process creates a subprocess OSCAR running the image SUBPROC.

❺ The interactive process hibernates.

❻ Because OSCAR is a subprocess, it can send output to a terminal allocated by its creator process. If OSCAR were a detached process, this would not be possible.

```
1       *                                                MAINPROC.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. MAINPROC.
5       *
6       *   This program illustrates process creation and
7       *   control.  It creates a subprocess and defines
8       *   SYS$OUTPUT, so that both processes can write
9       *   to the same terminal.
10      *
11      ENVIRONMENT DIVISION.
12      CONFIGURATION SECTION.
13      SPECIAL-NAMES.
14          SYMBOLIC CHARACTERS ESC NUL
15                          ARE  28   1.
16
17      DATA DIVISION.
18      WORKING-STORAGE SECTION.
19
20      01  LOGNAME          PIC X(255) VALUE 'SYS$OUTPUT'.
21      01  TERMNL           PIC X(255).
22      01  FILE_NAME        PIC X(7)  VALUE 'SUBPROC'.
23      01  MAIN_NAME        PIC X(8)  VALUE 'MAINPROC'.
24      01  SUB_NAME         PIC X(5)  VALUE 'OSCAR'.
25      01  TABLE_NAME       PIC X(17) VALUE 'LNM$PROCESS_TABLE'.
26      01  ESC-NULL.
27          05 ESC-CHAR      PIC X     VALUE ESC.
28          05 NULL-CHAR     PIC X     VALUE NUL.
29      01  PROCESS_ID       PIC 9(9)  COMP.
30      01  PID              PIC X(8).
31      01  LOGNAM_LEN       PIC 9(9)  COMP VALUE 10.
32      01  LOGNAM_LEN_LEN   PIC S9(4) COMP.
33      01  TERM_LEN         PIC S9(4) COMP.
34      01  STAT             PIC S9(9) COMP.
35      01  RET_ATTRIB       PIC 9(9) COMP.
36      01  ATT_LEN          PIC S9(4) COMP.
37      01  LNM$M_TERMINAL   PIC S9(9) COMP VALUE 512.
38      01  TRNLNM_LIST.
39          02  LNM_STRING.
40              03      PIC S9(4)       COMP VALUE 255.
41              03      PIC S9(4)       COMP VALUE 2.
42              03      POINTER VALUE REFERENCE TERMNL.
43              03      POINTER VALUE REFERENCE TERM_LEN.
44          02  LNM_ATTRIBUTES.
45              03      PIC S9(4)       COMP VALUE 4.
46              03      PIC S9(4)       COMP VALUE 3.
47              03      POINTER VALUE REFERENCE RET_ATTRIB.
48              03      POINTER VALUE REFERENCE ATT_LEN.
49          02  LNM_LENGTH.
50              03      PIC S9(4)       COMP VALUE 4.
51              03      PIC S9(4)       COMP VALUE 5.
52              03      POINTER VALUE REFERENCE LOGNAM_LEN.
53              03      POINTER VALUE REFERENCE LOGNAM_LEN_LEN.
54          02  TERMINATOR-ENTRY    PIC S9(9) COMP VALUE 0.
```

Example 3 Creating a Subprocess that Shares a Terminal With a Creating Process (Sheet 1 of 3)

```
55          *
56          PROCEDURE DIVISION.
57          BEGIN.
58          *
59          *   Translate SYS$OUTPUT
60      (1)     PERFORM TRANSLATE-LOGICAL-NAME WITH TEST AFTER
61                      UNTIL (LNM$M_TERMINAL + RET_ATTRIB) NOT = 0
62              IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
63          *
64          *   Check if process permanent file
65      (2)     IF TERMNL(1:2) IS EQUAL TO ESC-NULL
66                      MOVE TERMNL(5:TERM_LEN) TO TERMNL
67                      SUBTRACT 4 FROM TERM_LEN.
68          *
69          *   Set the process name
70      (3)     CALL 'SYS$SETPRN' USING BY DESCRIPTOR MAIN_NAME
71                               GIVING STAT.
72              IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
73          *
74          *   Create the subprocess
75      (4)     CALL 'SYS$CREPRC' USING BY REFERENCE PROCESS_ID
76                                      BY DESCRIPTOR FILE_NAME
77                                      BY VALUE 0
78                                      BY DESCRIPTOR TERMNL(1:TERM_LEN)
79                                      BY VALUE 0 0 0
80                                      BY DESCRIPTOR SUB_NAME
81                                      BY VALUE 4 0 0 0
82                                GIVING STAT.
83              IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
84          *
85          *   Convert process id to hexidecimal
86              CALL 'OTS$CVT_L_TZ' USING BY REFERENCE PROCESS_ID
87                                        BY DESCRIPTOR PID
88                                        BY VALUE 8 4
89                                 GIVING STAT.
90              IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
91              DISPLAY 'PID of subprocess OSCAR is ' PID.
92          *
93          *   Hibernate until awaken by subprocess
94      (5)     CALL 'SYS$HIBER' GIVING STAT.
95              IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
96              DISPLAY 'MAINPROC has been awakened.'.
97              STOP RUN.
98
99          TRANSLATE-LOGICAL-NAME.
100         *   Access contents of logical name
101             CALL 'SYS$TRNLNM' USING BY VALUE 0
102                                BY DESCRIPTOR TABLE_NAME LOGNAME(1:LOGNAM_LEN)
103                                BY VALUE 0
104                                BY REFERENCE TRNLNM_LIST
105                                  GIVING STAT.
106             IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
107             MOVE TERMNL TO LOGNAME.
```

Example 3 Creating a Subprocess that Shares a Terminal With a Creating Process (Sheet 2 of 3)

```
 1       *                                                SUBPROC.COB
 2       IDENTIFICATION DIVISION.
 3       *
 4       PROGRAM-ID. SUBPROC.
 5       *
 6       *   This program is run as a subprocess of program
 7       *   MAINPROC.  It issues a message to the device
 8       *   specified by the $CREPRC call, and awakens
 9       *   the main program.
10       *
11       DATA DIVISION.
12       WORKING-STORAGE SECTION.
13
14       01  STAT            PIC S9(9) COMP.
15       01  MAIN_NAME       PIC X(8)  VALUE 'MAINPROC'.
16
17       PROCEDURE DIVISION.
18       BEGIN.
19       *
20       *   Send a message to the terminal
21  (6)      DISPLAY 'This is from a subprocess.'
22       *
23       *   Wake the creator
24           CALL 'SYS$WAKE' USING BY VALUE 0
25                                 BY DESCRIPTOR MAIN_NAME
26                           GIVING STAT.
27           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
28           STOP RUN.
```

```
$ COBOL MAINPROC, SUBPROC
$ LINK MAINPROC
$ LINK SUBPROC
$ RUN MAINPROC
PID of subprocess OSCAR is 00000143
This is from a subprocess.
MAINPROC has been awakened.
$
```

Example 3 Creating a Subprocess that Shares a Terminal With a Creating Process (Sheet 3 of 3)

Example 4

Example 4 illustrates the ability of a created process to use the $GETJPIW system service to obtain the PID of its creator process.

❶ The subprocess is created using $CREPRC.

❷ The process hibernates.

❸ The item list for $GETJPIW consists of a single item descriptor followed by a zero longword.

❹ The EXTERNAL clause defines the value of the JPI$__OWNER code. JPI$__OWNER is the item code that requests the PID of the owner process. If there is no owner process (for example, if the information is requested about a detached process), the system service $GETJPIW returns a PID of zero.

❺ Because of the item code JPI$__OWNER in the item list, $GETJPIW returns the PID of the owner of the process about which information is requested. If the item code were JPI$__PID, $GETJPIW would return the PID of the process about which information is requested.

Because the default value of 0 is used for arguments PIDADR and PRCNAM, the process about which information is requested is the requesting process, namely, OSCAR.

```
1       *                                                GETJPI.COB
2       *
3       IDENTIFICATION DIVISION.
4       *
5       PROGRAM-ID. GETJPI.
6       *
7       *   This program illustrates process creation and
8       *   control.  It creates a subprocess then hibernates
9       *   until the subprocess wakes.
10      *
11      ENVIRONMENT DIVISION.
12      CONFIGURATION SECTION.
13      SPECIAL-NAMES.
14          SYMBOLIC CHARACTERS ESC NUL
15                              ARE  28   1.
16
17      DATA DIVISION.
18      WORKING-STORAGE SECTION.
19
20      01  LOGNAME          PIC X(255) VALUE 'SYS$OUTPUT'.
21      01  TERMNAL          PIC X(255).
22      01  FILE_NAME                PIC X(9)  VALUE 'GETJPISUB'.
23      01  SUB_NAME                 PIC X(5)  VALUE 'OSCAR'.
24      01  ESC-NULL.
25          05      ESC-CHAR          PIC X     VALUE ESC.
26          05      NULL-CHAR         PIC X     VALUE NUL.
27      01  PROCESS_ID               PIC 9(9)  COMP.
28      01  PID                      PIC X(8).
29      01  LOGNAME_LEN              PIC 9(9)  COMP VALUE 10.
30      01  TERM_LEN                 PIC S9(4)  COMP VALUE 255.
31      01  RET_ATTRIB               PIC 9(9) COMP.
32      01  ATT_LEN          PIC S9(4) COMP.
33      01  STAT             PIC S9(9) COMP.
34      01  TABLE_NAME               PIC X(17) VALUE 'LNM$PROCESS_TABLE'.
35      01  RET_LENGTH               PIC S9(9) COMP VALUE 10.
36      01  RET_LENGTH_LEN   PIC S9(4) COMP.
37      01  LNM$M_TERMINAL   PIC S9(9) COMP VALUE 512.
38      01  TRNLNM_LIST.
39          02  LNM_STRING.
40          03      PIC S9(4)         COMP VALUE 255.
41          03      PIC S9(4)         COMP VALUE 2.
42          03      POINTER VALUE REFERENCE TERMNAL.
43          03      POINTER VALUE REFERENCE TERM_LEN.
44          02  LNM_LENGTH.
45          03      PIC S9(4)         COMP VALUE 4.
46          03      PIC S9(4)         COMP VALUE 5.
47          03      POINTER VALUE REFERENCE RET_LENGTH.
48          03      POINTER VALUE REFERENCE RET_LENGTH_LEN.
49          02  LNM_ATTRIBUTES.
50          03      PIC S9(4)         COMP VALUE 4.
51          03      PIC S9(4)         COMP VALUE 3.
52          03      POINTER VALUE REFERENCE RET_ATTRIB.
53          03      POINTER VALUE REFERENCE ATT_LEN.
54          02  TERMINATOR-ENTRY     PIC S9(9) COMP VALUE 0.
```

Example 4 Using the $GETJPIW System Service to Obtain the Creator's PID (Sheet 1 of 3)

```
55        *
56        PROCEDURE DIVISION.
57        BEGIN.
58        *
59        *   Translate SYS$OUTPUT
60            PERFORM TRANSLATE-LOGICAL-NAME WITH TEST AFTER
61                    UNTIL (LNM$M_TERMINAL + RET_ATTRIB) NOT = 0
62        *
63        *   Check if process permanent file
64            IF TERMNAL(1:2) IS EQUAL TO ESC-NULL
65                    MOVE TERMNAL(5:TERM_LEN) TO TERMNAL
66                    SUBTRACT 4 FROM TERM_LEN.
67        *
68        *   Create the subprocess
69    (1)     CALL 'SYS$CREPRC' USING BY REFERENCE PROCESS_ID
70                                    BY DESCRIPTOR FILE_NAME
71                                    BY VALUE 0
72                                    BY DESCRIPTOR TERMNAL(1:TERM_LEN)
73                                    BY VALUE 0 0 0
74                                    BY DESCRIPTOR SUB_NAME
75                                    BY VALUE 4 0 0 0
76                              GIVING STAT.
77            IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
78        *
79        *   Convert process id to hexidecimal
80            CALL 'OTS$CVT_L_TZ' USING BY REFERENCE PROCESS_ID
81                                      BY DESCRIPTOR PID
82                                      BY VALUE 8 4
83                                GIVING STAT.
84            IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
85            DISPLAY 'PID of subprocess OSCAR is ' PID.
86        *
87        *   Hibernate until awaken by subprocess
88    (2)     CALL 'SYS$HIBER' GIVING STAT.
89            IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
90            DISPLAY 'GETJPI has been awakened.'.
91            STOP RUN.
92
93        TRANSLATE-LOGICAL-NAME.
94        *   Access contents of supervisor mode logical name
95            CALL 'SYS$TRNLNM' USING BY VALUE 0
96                              BY DESCRIPTOR TABLE_NAME LOGNAME(1:RET_LENGTH)
97                              BY VALUE 0
98                              BY REFERENCE TRNLNM_LIST
99                                 GIVING STAT.
100           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
101           MOVE TERMNAL TO LOGNAME.
```

Example 4 Using the $GETJPIW System Service to Obtain the Creator's PID (Sheet 2 of 3)

```
 1       *                                                GETJPISUB.COB
 2       *
 3       IDENTIFICATION DIVISION.
 4       *
 5       PROGRAM-ID. GETJPISUB.
 6       *
 7       *   This program is run in the subprocess OSCAR
 8       *   which is created by GETJPI.  It obtains its
 9       *   creator's PID then wakes it.
10       *
11       DATA DIVISION.
12       WORKING-STORAGE SECTION.
13
14    3  01  JPI_LIST.
15          02  ITEM-OWNER-PID.
16            03     PIC S9(4)       COMP VALUE IS 4.
17    4       03     PIC S9(4)       COMP VALUE EXTERNAL JPI$_OWNER.
18            03 POINTER VALUE REFERENCE OWNER_PID.
19            03 POINTER VALUE REFERENCE OWNER_PID_LENGTH.
20          02  TERMINATOR-ENTRY     PIC S9(9) COMP  VALUE 0.
21       01 OWNER_PID                PIC S9(9) COMP.
22       01 OWNER_PID_LENGTH PIC S9(4) COMP.
23       01 STAT                     PIC S9(9) COMP.
24       01 IOSB.
25           02  IOSB1               PIC S9(9) COMP.
26           02  IOSB2               PIC S9(9) COMP.
27
28       PROCEDURE DIVISION.
29       BEGIN.
30       *
31       *   Get PID of creator
32    5      CALL 'SYS$GETJPIW' USING BY VALUE 1 0 0
33                                    BY REFERENCE JPI_LIST IOSB
34                                    BY VALUE 0 0
35                              GIVING STAT.
36           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
37           IF IOSB1 IS FAILURE CALL 'LIB$STOP' USING BY VALUE IOSB1.
38       *
39       *   Wake creator
40           DISPLAY 'OSCAR is waking creator.'.
41           CALL 'SYS$WAKE' USING BY REFERENCE OWNER_PID
42                                 BY VALUE 0
43                           GIVING STAT.
44           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
45           STOP RUN.
```

```
$ COBOL GETJPI, GETJPISUB
$ LINK GETJPI
$ LINK GETJPISUB
$ RUN GETJPI
PID of subprocess OSCAR is 0000012B
OSCAR is waking creator.
GETJPI has been awakened.
$
```

Example 4 Using the $GETJPIW System Service to Obtain the Creator's PID (Sheet 3 of 3)

Lab Exercises

1. Write a program (LABSOL1.COB) that uses system services $ASCEF and $SETEF to associate to a common event flag cluster called MS__PIGGY and set event flag 70.

2. Write a program (LABSOL2.COB) that does the following:
   a. Uses the $CREPRC system service to create a subprocess called OSCAR that runs LABSOL1
   b. Waits for the subprocess to set event flag 70
   c. Issues a message that event flag 70 has been set

3. Modify LABSOL1 and LABSOL2 as follows:
   a. The main program (LABSOL2) should:
      1) Create a hibernating subprocess (LABSOL1).
      2) Schedule the subprocess to be awakened at half-second intervals.
      3) Wait for the subprocess to set event flag 70, then cancel further scheduled wakeups for the subprocess and force it to exit.
      4) Issue messages that wakeup for OSCAR has been scheduled and that event flag 70 has been set.
   b. The subprogram (LABSOL1) should:
      1) Declare an integer variable COUNTDOWN.
      2) Increase COUNTDOWN by one each time the subprocess is awakened.
      3) After being awakened five times, the subprocess should set event flag 70.

Lab Solutions

1.

```
1       *                                                    LABSOL1.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL1.
5       *
6       *   This program associates to a common event flag
7       *   cluster called MS_PIGGY and sets event flag
8       *   number 70.
9       *
10      DATA DIVISION.
11      WORKING-STORAGE SECTION.
12
13      01  CLUSTER_NAME    PIC X(8)        VALUE 'MS_PIGGY'.
14      01  STAT            PIC S9(9) COMP.
15
16      PROCEDURE DIVISION.
17      BEGIN.
18      *
19      *   Associate to MS_PIGGY and set event flag 70
20          CALL 'SYS$ASCEFC' USING BY VALUE 70
21                                  BY DESCRIPTOR CLUSTER_NAME
22                                  BY VALUE 0 0
23                            GIVING STAT.
24          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
25          CALL 'SYS$SETEF' USING BY VALUE 70
26                           GIVING STAT.
27          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT
28          EXIT PROGRAM.
```

2.

```
1       *                                                    LABSOL2.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL2.
5       *
6       *   This process:
7       *     1. Creates a subprocess that runs LABSOL1
8       *     2. Waits for the subprocess to set event flag 70
9       *     3. Issues a message that event flag 70 has been set.
10      *
11      DATA DIVISION.
12      WORKING-STORAGE SECTION.
13
14      01  PROCESS_ID      PIC S9(9)  COMP VALUE 0.
15      01  FILE_NAME       PIC X(7)        VALUE 'LABSOL1'.
16      01  SUB_NAME        PIC X(5)        VALUE 'OSCAR'.
17      01  CLUSTER_NAME    PIC X(8)        VALUE 'MS_PIGGY'.
18      01  STAT            PIC S9(9)  COMP.
19
20      PROCEDURE DIVISION.
21      BEGIN.
22      *
23      *   Create the subprocess
24          CALL 'SYS$CREPRC' USING BY REFERENCE PROCESS_ID
25                                  BY DESCRIPTOR FILE_NAME
26                                  BY VALUE 0 0 0 0 0
27                                  BY DESCRIPTOR SUB_NAME
28                                  BY VALUE 4 0 0 0
29                            GIVING STAT.
30          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
31      *
32      *   Associate to cluster MS_PIGGY and wait for event
33      *   flag 70
34          CALL 'SYS$ASCEFC' USING BY VALUE 70
35                                  BY DESCRIPTOR CLUSTER_NAME
36                                  BY VALUE 0 0
37                            GIVING STAT.
38          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
39          CALL 'SYS$WAITFR' USING BY VALUE 70
40                            GIVING STAT.
41          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
42          DISPLAY 'EFN 70 has been set.'.
43      *
44          STOP RUN.
```

3.

```
1       *                                                LABSOL3A.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL3A.
5       *
6       *   This process:
7       *     1. Creates a hibernating subprocess
8       *     2. Schedules the subprocess to be awakened at
9       *           half second intervals
10      *     3. Waits for the subprocess to set event flag 70
11      *     4. Cancels further scheduled wake-ups for the
12      *           subprocess
13      *     5. Forces the subprocess to exit
14      *
15      DATA DIVISION.
16      WORKING-STORAGE SECTION.
17
18      01  CURRENT_TIME    PIC S9(18) COMP.
19      01  INTERVAL        PIC S9(18) COMP.
20      01  PROCESS_ID      PIC S9(9)  COMP VALUE 0.
21      01  HIBER_FLAG      PIC S9(9)  COMP VALUE 32.
22      01  FILE_NAME       PIC X(8)        VALUE 'LABSOL3B'.
23      01  SUB_NAME        PIC X(5)        VALUE 'OSCAR'.
24      01  CLUSTER_NAME    PIC X(8)        VALUE 'MS_PIGGY'.
25      01  ASCII_TIME      PIC X(7)        VALUE '0 ::.50'.
26      01  STAT            PIC S9(9)  COMP.
27
28      PROCEDURE DIVISION.
29      BEGIN.
30      *
31      *   Create the subprocess
32          CALL 'SYS$CREPRC' USING BY REFERENCE PROCESS_ID
33                                  BY DESCRIPTOR FILE_NAME
34                                  BY VALUE 0 0 0 0 0
35                                  BY DESCRIPTOR SUB_NAME
36                                  BY VALUE 4 0 0 HIBER_FLAG
37                           GIVING STAT.
38          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
39      *
40      *   Reschedule the subprocess to wakeup every .5 second
41          DISPLAY 'Scheduling wakeup for OSCAR.'
42          CALL 'SYS$BINTIM' USING BY DESCRIPTOR ASCII_TIME
43                                  BY REFERENCE INTERVAL.
44          CALL 'SYS$GETTIM' USING BY REFERENCE CURRENT_TIME.
45          CALL 'SYS$SCHDWK' USING BY REFERENCE PROCESS_ID
46                                  BY VALUE 0
47                                  BY REFERENCE CURRENT_TIME INTERVAL
48                           GIVING STAT.
49          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
50      *
```

(Sheet 1 of 2)

```
51          *   Associate to cluster MS_PIGGY and wait for event
52          *   flag 70
53              CALL 'SYS$ASCEFC' USING BY VALUE 70
54                                      BY DESCRIPTOR CLUSTER_NAME
55                                      BY VALUE 0 0
56                               GIVING STAT.
57              IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
58              CALL 'SYS$WAITFR' USING BY VALUE 70
59                               GIVING STAT.
60              IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
61              DISPLAY 'EFN 70 has been set.'.
62          *
63          *   Now cancel the scheduled wake-up for the subprocess
64              CALL 'SYS$CANWAK' USING BY REFERENCE PROCESS_ID
65                                      BY VALUE 0
66                               GIVING STAT.
67              IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
68          *
69          *   And force it to exit
70              CALL 'SYS$FORCEX' USING BY REFERENCE PROCESS_ID
71                                      BY VALUE 0 0
72                               GIVING STAT.
73              IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
74              STOP RUN.
```

```
 1          *                                                LABSOL3B.COB
 2          IDENTIFICATION DIVISION.
 3          *
 4          PROGRAM-ID. LABSOL3B.
 5          *
 6          *   This subprocess is requested by LABSOL3A with a
 7          *   reschedule interval of half a second. It is created
 8          *   in a hibernating state. After being awakened 5
 9          *   times this process will set event flag 70.
10          *
11          DATA DIVISION.
12          WORKING-STORAGE SECTION.
13
14          01  COUNTDOWN       PIC S9(9) COMP  VALUE 5.
15          01  CLUSTER_NAME    PIC X(8)        VALUE 'MS_PIGGY'.
16          01  STAT            PIC S9(9) COMP.
17
18          PROCEDURE DIVISION.
19          BEGIN.
20          *
21          *   Set event flag after being awakened 5 times
22              SUBTRACT 1 FROM COUNTDOWN
23
24              IF COUNTDOWN IS EQUAL TO 0 THEN
25          *           Associate to MS_PIGGY and set event flag 70
26                  CALL 'SYS$ASCEFC' USING BY VALUE 70
27                                          BY DESCRIPTOR CLUSTER_NAME
28                                          BY VALUE 0 0
29                                   GIVING STAT
30                  IF STAT IS FAILURE
31                          CALL 'LIB$STOP' USING BY VALUE STAT
32                  END-IF
33                  CALL 'SYS$SETEF' USING BY VALUE 70
34                                   GIVING STAT
35                  IF STAT IS FAILURE
36                          CALL 'LIB$STOP' USING BY VALUE STAT
37                  END-IF
38              END-IF
39              EXIT PROGRAM.
```

(Sheet 2 of 2)

Gathering and Displaying Data at Terminals

File Locations

On-line files are provided to help you master this module. The files are located in the directory:

```
DISK$COURSE:[COURSE.V4PROG.COB.TERM]
```

For your convenience, your system manager may have created the following logical name equivalence:

```
DISK$COURSE:[COURSE.V4PROG.COB.TERM] = V4PROG$COB$TERM
```

Three types of files are provided:

1. The Module Programming Examples, which are also listed in this language workbook.
2. Files that you modify to complete the Laboratory Exercises.
3. Solutions to the Laboratory Exercises.

If you wish to modify and/or compile, link, and run any of the programs, first copy the file into your directory.

Example 1

Example 1 calls SMG routines to format screen output.

1. The call to SMG$CREATE_PASTEBOARD creates a pasteboard on which output will be written. The pasteboard ID is returned in the variable NEW_PID.

   No value is specified for the output device parameter, so the output device defaults to SYS$OUTPUT. Also, no values are specified for the PB_ROWS or PB_COLS parameters, so the pasteboard is created with the default number of rows and columns. The defaults are the number of rows and the number of columns on the physical screen of the terminal to which SYS$OUTPUT is assigned.

2. The created virtual display is 15 lines long and 30 columns wide; it initially contains blanks.

3. The virtual display is pasted to the pasteboard, with the upper left corner positioned at row 2, column 15 of the pasteboard. Pasting the virtual display to the pasteboard causes all data written to the virtual display to appear on the pasteboard's output device, which is SYS$OUTPUT—the terminal screen.

   At this point, nothing will appear on the terminal screen because the virtual display contains only blanks. However, because the virtual display is pasted to the pasteboard, the program statements described below cause text to be written to the screen.

4. A labeled border is written to the virtual display.

5. Text lines are written to the virtual display. The LINE_ADV parameter specifies double spacing.

6. These statements use RENDITION_SET and RENDITION_COMPLEMENT parameters to display blinking and reverse video text.

7. Single spaced text is displayed.

No sample run is included for this example because the program requires a video terminal to execute properly.

LEARNING ACTIVITY

After running this program, move the call to SMG$PASTE_VIRTUAL_DISPLAY to the end of the program and observe the results.

```
1       *                                                SMGOUTPUT.COB
2       *
3       IDENTIFICATION DIVISION.
4       *
5       PROGRAM-ID. SMGOUTPUT.
6       *
7       *   This program calls Run-Time Library Screen Management
8       *   routines to format screen output.
9       *
10      DATA DIVISION.
11      WORKING-STORAGE SECTION.
12
13      01  NEW_PID          PIC S9(9) COMP.
14      01  DISPLAY_ID       PIC S9(9) COMP.
15      01  STAT             PIC S9(9) COMP.
16      01  SMG$M_REVERSE    PIC S9(9) COMP VALUE 2.
17      01  SMG$M_BLINK      PIC S9(9) COMP VALUE 4.
18      01  SMG$M_UNDERLINE PIC S9(9) COMP VALUE 8.
19      01  SMG$M_BORDER     PIC S9(9) COMP VALUE 1.
20      01  LOOP_CTR         PIC S9(9) COMP VALUE 1.
21      01  DISP_ROWS        PIC S9(9) COMP VALUE 15.
22      01  DISP_COLS        PIC S9(9) COMP VALUE 30.
23      01  PASTE_ROW        PIC S9(9) COMP VALUE 2.
24      01  PASTE_COL        PIC S9(9) COMP VALUE 30.
25      01  DBL_SPC          PIC S9(9) COMP VALUE 2.
26      01  BORD_LABL        PIC X(18) VALUE 'This is the Border'.
27
28      PROCEDURE DIVISION.
29      BEGIN.
30
31      *   establish terminal screen as pasteboard
32  (1)     CALL 'SMG$CREATE_PASTEBOARD' USING BY REFERENCE NEW_PID
33                       BY VALUE 0 0 0
34                       GIVING STAT.
35          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
36      *
37      *   establish a virtual display region
38  (2)     CALL 'SMG$CREATE_VIRTUAL_DISPLAY' USING
39                       BY REFERENCE DISP_ROWS DISP_COLS
40                       BY REFERENCE DISPLAY_ID
41                       BY VALUE 0 0 0 GIVING STAT.
42          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
43      *
44      *   paste the virtual display to the screen, starting at
45      *   row 2, column 15
46  (3)     CALL 'SMG$PASTE_VIRTUAL_DISPLAY' USING
47                       BY REFERENCE DISPLAY_ID NEW_PID
48                       PASTE_ROW PASTE_COL GIVING STAT.
49          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
```

Example 1 Using SMG$ Procedures to Send Output to the Terminal (Sheet 1 of 2)

```
$
```

```
50          *
51          *   Put a border around the display area
52      4       CALL 'SMG$LABEL_BORDER' USING
53                              BY REFERENCE DISPLAY_ID
54                              BY DESCRIPTOR BORD_LABL
55                              BY VALUE 0 0 0 0 0
56                              GIVING STAT.
57              IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
58          *
59          *   write text lines to the screen
60          *
61              CALL 'SMG$PUT_LINE' USING
62                              BY REFERENCE DISPLAY_ID
63                              BY DESCRIPTOR ' '
64                              BY VALUE 0 0 0 0 0
65                              GIVING STAT.
66              IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
67              CALL 'SMG$PUT_LINE' USING
68                              BY REFERENCE DISPLAY_ID
69                              BY DESCRIPTOR 'Howdy, Pardner'
70                              BY REFERENCE DBL_SPC
71      5                       BY VALUE 0 0 0 0
72                              GIVING STAT.
73              IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
74              CALL 'SMG$PUT_LINE' USING
75                              BY REFERENCE DISPLAY_ID
76                              BY DESCRIPTOR 'Double Spaced lines...'
77                              BY REFERENCE DBL_SPC
78                              BY VALUE 0 0 0 0
79                              GIVING STAT.
80              IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
81              CALL 'SMG$PUT_LINE' USING
82                              BY REFERENCE DISPLAY_ID
83                              BY DESCRIPTOR 'This line is blinking'
84                              BY REFERENCE DBL_SPC SMG$M_BLINK
85                              BY VALUE 0 0 0
86                              GIVING STAT.
87      6       IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
88              CALL 'SMG$PUT_LINE' USING
89                              BY REFERENCE DISPLAY_ID
90                              BY DESCRIPTOR 'This line is reverse video'
91                              BY REFERENCE DBL_SPC SMG$M_REVERSE
92                              BY VALUE 0 0 0
93                              GIVING STAT.
94              IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
95          *
96              PERFORM UNTIL LOOP_CTR = 5
97                      CALL 'SMG$PUT_LINE' USING BY REFERENCE DISPLAY_ID
98                                      BY DESCRIPTOR 'Single spaced lines...'
99                                      BY VALUE 0 0 0 0 0
100     7                               GIVING STAT
101                     IF STAT IS FAILURE
102                             CALL 'LIB$STOP' USING BY VALUE STAT
103                     END-IF
104                     ADD 1 TO LOOP_CTR
105             END-PERFORM.
106             STOP RUN.
```

Example 1 Using SMG$ Procedures to Send Output to the Terminal (Sheet 2 of 2)

Example 2

Example 2 calls the SMG routines to format screen output and accept user input.

❶ A virtual keyboard creates an association between a terminal (or file) and a virtual keyboard identifier. No value is specified for the FILESPEC parameter, so SYS$INPUT (the terminal) is the default.

❷ As in Example 1, a pasteboard and a virtual display are created, and the virtual display is pasted to the pasteboard.

❸ A menu is displayed, double-spaced.

❹ This statement writes a prompt on the virtual display (it is displayed on the terminal), and accepts user input from the virtual keyboard associated with SYS$INPUT.

❺ The menu only offers two selections. If the user enters an invalid selection number, the screen is erased and the menu redisplayed.

❻ If the user chooses selection 1 from the menu, the video rendition of the screen is changed to reverse video.

❼ If the user chooses selection 2 from the menu, a new virtual display is created and pasted to the pasteboard. The call to SMG$SET_DISPLAY_SCROLL_REGION defines the virtual display to be a scrolling region.

❽ The ten array elements are displayed in the four-line scrolling region. SMG$M_UP is specified as the DIRECTION parameter. Consequently, as each array element is displayed, one previously displayed element is scrolled up, out of the display area.

❾ The ten array elements are again displayed, this time in reverse order. SMG$M_DOWN is specified as the DIRECTION parameter. Consequently, as each array element is displayed, one previously displayed element is scrolled down, out of the four-line display area.

No sample run is included for this example because the program requires a video terminal to execute properly.

```
1       *                                                SMGMENU.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. SMGMENU.
5       *
6       *   This program calls Run-Time Library Screen Management
7       *   routines to format screen output and accept user input.
8       *   It will produce a high-wide display, or a scrolled display.
9       *
10      DATA DIVISION.
11      WORKING-STORAGE SECTION.
12
13      01  SMG$M_UP          PIC S9(9) COMP VALUE 1.
14      01  SMG$M_DOWN        PIC S9(9) COMP VALUE 2.
15      01  SMG$M_BOLD        PIC S9(9) COMP VALUE 1.
16      01  SMG$M_REVERSE     PIC S9(9) COMP VALUE 2.
17      01  SMG$M_BLINK       PIC S9(9) COMP VALUE 4.
18      01  SMG$M_UNDERLINE PIC S9(9) COMP VALUE 8.
19      01  OUT_STR           PIC X.
20      01  PROMPT      PIC X(32) VALUE 'Enter the number of your choice '.
21      01  ARRAY.
22          02  ELEMNT        PIC X(7) OCCURS 10 TIMES.
23      01  DISPLAY_ID        PIC S9(9) COMP.
24      01  KEYBOARD_ID       PIC S9(9) COMP.
25      01  NEW_PID           PIC S9(9) COMP.
26      01  SCRL_ID           PIC S9(9) COMP.
27      01  STAT              PIC S9(9) COMP.
28      01  IDX               PIC S9(9) COMP.
29      01  DISP_ROWS         PIC S9(9) COMP VALUE 20.
30      01  DISP_COLS         PIC S9(9) COMP VALUE 65.
31      01  PASTE_ROW         PIC S9(9) COMP VALUE 3.
32      01  PASTE_COL         PIC S9(9) COMP VALUE 15.
33      01  SCRL_ROWS         PIC S9(9) COMP VALUE 4.
34      01  SCRL_COLS         PIC S9(9) COMP VALUE 65.
35      01  DBL_SPC           PIC S9(9) COMP VALUE 2.
36      01  N_ROW             PIC S9(9) COMP.
37      01  N_COL             PIC S9(9) COMP.
38
39      PROCEDURE DIVISION.
40      BEGIN.
41
42      *   establish terminal keyboard as virtual keyboard
43  (1)     CALL 'SMG$CREATE_VIRTUAL_KEYBOARD' USING
44                          BY REFERENCE KEYBOARD_ID
45                          BY VALUE 0 0 0 GIVING STAT.
46          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
47
48      *   establish terminal screen as a pasteboard
49          CALL 'SMG$CREATE_PASTEBOARD' USING
50  (2)                     BY REFERENCE NEW_PID
51                          BY VALUE 0 0 0 GIVING STAT.
52          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
53
54      *   establish a virtual display region
55          CALL 'SMG$CREATE_VIRTUAL_DISPLAY' USING
56                          BY REFERENCE DISP_ROWS DISP_COLS
57                          BY REFERENCE DISPLAY_ID
58                          BY VALUE 0 0 0 GIVING STAT.
59          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
60  (2)
61      *   paste the virtual display to the screen, starting at
62      *   row 3, column 15
63          CALL 'SMG$PASTE_VIRTUAL_DISPLAY' USING
64                          BY REFERENCE DISPLAY_ID NEW_PID
65                          BY REFERENCE PASTE_ROW PASTE_COL GIVING STAT.
66          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
67
```

Example 2 Using SMG$ Procedures to Accept Input and Send Output to the Terminal (Sheet 1 of 3)

```
68      DISPLAY-MENU.
69
70      *   display the menu selctions
71          CALL 'SMG$PUT_LINE' USING
72                          BY REFERENCE DISPLAY_ID
73                          BY DESCRIPTOR ' '
74                          BY REFERENCE DBL_SPC
75                          BY VALUE 0 0 0 0 GIVING STAT.
76          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
77          CALL 'SMG$PUT_LINE' USING
78                          BY REFERENCE DISPLAY_ID
79      3                   BY DESCRIPTOR '1.  Change Video Rendition'
80                          BY REFERENCE DBL_SPC
81                          BY VALUE 0 0 0 0 GIVING STAT.
82          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
83          CALL 'SMG$PUT_LINE' USING
84                          BY REFERENCE DISPLAY_ID
85                          BY DESCRIPTOR '2.  Scrolled Display'
86                          BY REFERENCE DBL_SPC
87                          BY VALUE 0 0 0 0 GIVING STAT.
88          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
89
90      *   prompt user and accept input
91          CALL 'SMG$READ_STRING' USING
92                          BY REFERENCE KEYBOARD_ID
93      4                   BY DESCRIPTOR OUT_STR PROMPT
94                          BY VALUE 0 0 0 0 0 0
95                          BY REFERENCE DISPLAY_ID GIVING STAT.
96          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
97
98      *   test for validity of user input
99          IF ((OUT_STR NOT = '1') AND (OUT_STR NOT = '2')) THEN
100     *           erase the display, so redisplay the menu
101                 CALL 'SMG$ERASE_DISPLAY' USING
102                         BY REFERENCE DISPLAY_ID
103     5                   BY VALUE 0 0 0 0 GIVING STAT
104                 IF STAT IS FAILURE THEN
105                         CALL 'LIB$STOP' USING BY VALUE STAT
106                 END-IF
107                 GO TO DISPLAY-MENU
108         END-IF.
109     *
110     *   display the users selection
111         IF (OUT_STR = '1') THEN
112                 MOVE 2 TO N_ROW
113                 MOVE -32 TO N_COL
114                 CALL 'SMG$SET_CURSOR_REL' USING
115                         BY REFERENCE DISPLAY_ID
116                         BY REFERENCE N_ROW N_COL GIVING STAT
117                 IF STAT IS FAILURE THEN
118                         CALL 'LIB$STOP' USING BY VALUE STAT
119                 END-IF
120                 CALL 'SMG$PUT_CHARS' USING
121                           BY REFERENCE DISPLAY_ID
122                           BY DESCRIPTOR 'Changing video rendition...'
123                           BY VALUE 0 0 0 0 0 GIVING STAT
124     6           IF STAT IS FAILURE THEN
125                         CALL 'LIB$STOP' USING BY VALUE STAT
126                 END-IF
127                 MOVE 1 TO N_ROW
128                 MOVE 1 TO N_COL
129                 CALL 'SMG$CHANGE_RENDITION' USING
130                               BY REFERENCE DISPLAY_ID N_ROW N_COL
131                               BY REFERENCE DISP_ROWS DISP_COLS
132                               BY REFERENCE SMG$M_REVERSE
133                               BY VALUE 0 GIVING STAT
134                 IF STAT IS FAILURE THEN
135                         CALL 'LIB$STOP' USING BY VALUE STAT
136                 END-IF
137         END-IF.
```

Example 2 Using SMG$ Procedures to Accept Input and Send Output to the Terminal (Sheet 2 of 3)

```
138             IF (OUT_STR = '2') THEN
139         *       define a scrolling region
140                 CALL 'SMG$CREATE_VIRTUAL_DISPLAY' USING
141                           BY REFERENCE SCRL_ROWS SCRL_COLS SCRL_ID
142                           BY VALUE 0 0 0 GIVING STAT
143     (7)         IF STAT IS FAILURE THEN
144                     CALL 'LIB$STOP' USING BY VALUE STAT
145                 END-IF
146                 MOVE 12 TO N_ROW
147                 MOVE 15 TO N_COL
148                 CALL 'SMG$PASTE_VIRTUAL_DISPLAY' USING
149                               BY REFERENCE SCRL_ID NEW_PID
150                               BY REFERENCE N_ROW N_COL GIVING STAT
151                 IF STAT IS FAILURE THEN
152     (7)             CALL 'LIB$STOP' USING BY VALUE STAT
153                 END-IF
154                 CALL 'SMG$SET_DISPLAY_SCROLL_REGION' USING
155                               BY REFERENCE SCRL_ID
156                               BY VALUE 0 0 GIVING STAT
157                 IF STAT IS FAILURE THEN
158                     CALL 'LIB$STOP' USING BY VALUE STAT
159                 END-IF
160
161         *       fill the array
162                 MOVE 'First' TO ELEMNT(1)
163                 MOVE 'Second' TO ELEMNT(2)
164                 MOVE 'Third' TO ELEMNT(3)
165                 MOVE 'Fourth' TO ELEMNT(4)
166                 MOVE 'Fifth' TO ELEMNT(5)
167                 MOVE 'Sixth' TO ELEMNT(6)
168                 MOVE 'Seventh' TO ELEMNT(7)
169                 MOVE 'Eighth' TO ELEMNT(8)
170                 MOVE 'Ninth' TO ELEMNT(9)
171                 MOVE 'Tenth' TO ELEMNT(10)
172
173         *       display the ARRAY elements in the scrolling region
174                 PERFORM VARYING IDX FROM 1 BY 1 UNTIL IDX EQUAL 10
175                         CALL 'SMG$PUT_WITH_SCROLL' USING
176                                   BY REFERENCE SCRL_ID
177                                   BY DESCRIPTOR ELEMNT(IDX)
178     (8)                           BY REFERENCE SMG$M_UP
179                                   BY VALUE 0 0 0 0 GIVING STAT
180                         IF STAT IS FAILURE THEN
181                                   CALL 'LIB$STOP' USING BY VALUE STAT
182                         END-IF
183                 END-PERFORM
184                 PERFORM VARYING IDX FROM 10 BY -1 UNTIL IDX EQUAL 1
185                         CALL 'SMG$PUT_WITH_SCROLL' USING
186                                   BY REFERENCE SCRL_ID
187                                   BY DESCRIPTOR ELEMNT(IDX)
188     (9)                           BY REFERENCE SMG$M_DOWN
189                                   BY VALUE 0 0 0 0 GIVING STAT
190                         IF STAT IS FAILURE THEN
191                                   CALL 'LIB$STOP' USING BY VALUE STAT
192                         END-IF
193                 END-PERFORM
194             END-IF.
195             STOP RUN.
$
```

Example 2 Using SMG$ Procedures to Accept Input and Send Output to the Terminal (Sheet 3 of 3)

Example 3

This program illustrates the use of the $QIO system service to communicate with the terminal, and to respond to a CTRL/C typed by a user. It also illustrates the use of function code modifiers.

❶ Before any $QIO can be issued, a channel must be assigned to the device.

❷ A CTRL/C AST routine is declared. Note that this routine will only be entered for the first CTRL/C the user types.

❸ This request prompts a user for input. It sets up a time limit to wait between characters typed (five seconds). Also, any characters typed will not be echoed on the terminal. An I/O status block is established to contain the final I/O status and byte count for characters read.

❹ Specific tests are made for CTRL/C or timeout detection using the final I/O status in the I/O status block.

❺ If the user manages to enter a line within the time limit, it is displayed. The byte count in the I/O status block is used to determine the length of the user data input.

❻ The CTRL/C routine cancels any outstanding requests on the terminal channel.

❼ Before exiting, the CTRL/C AST routine issues a message to the user, indicating that the AST routine was called.

The sample runs illustrate the three possibilities that may occur when running this program (typing a CTRL/C, entering data, and not responding within the time limit).

```
1       *                                                CONTROLC.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. CONTROLC.
5       *
6       *   This program illustrates the use of QIOs to the terminal
7       *   and establishing an AST routine to handle control-Cs.
8       *
9       DATA DIVISION.
10      WORKING-STORAGE SECTION.
11
12      01  STATUS-BLOCK.
13          03      IOSB-STATUS     PIC S9(4)       COMP.
14          03      IOSB-COUNT      PIC S9(4)       COMP.
15          03      IOSB-INFO       PIC S9(4)       COMP.
16      01  STAT            PIC S9(9)   COMP.
17      01  TERM            PIC X(11)   VALUE 'SYS$COMMAND'.
18      01  BUFFER          PIC X(10).
19      01  PROMPT          PIC X(48)   VALUE
20          'You have 5 seconds to enter data, or type ctrl-C'.
21      01  TTCHAN          PIC S9(9) COMP.
22      01  IO$_SETMODE     PIC S9(9) COMP  VALUE EXTERNAL IO$_SETMODE.
23      01  IO$M_CTRLCAST   PIC S9(9) COMP  VALUE EXTERNAL IO$M_CTRLCAST.
24      01  IO$_READPROMPT  PIC S9(9) COMP  VALUE EXTERNAL IO$_READPROMPT.
25      01  IO$M_NOECHO     PIC S9(9) COMP  VALUE EXTERNAL IO$M_NOECHO.
26      01  IO$M_TIMED      PIC S9(9) COMP  VALUE EXTERNAL IO$M_TIMED.
27      01  SS$_NORMAL      PIC S9(9) COMP  VALUE EXTERNAL SS$_NORMAL.
28      01  SS$_CONTROLC    PIC S9(9) COMP  VALUE EXTERNAL SS$_CONTROLC.
29      01  SS$_TIMEOUT     PIC S9(9) COMP  VALUE EXTERNAL SS$_TIMEOUT.
30      01  AST-PROC        PIC S9(9) COMP  VALUE EXTERNAL AST-PROC.
31      01  FUNC            PIC S9(9) COMP.
32
33      PROCEDURE DIVISION.
34      BEGIN.
35      *
36      *   Assign channel to terminal
37  (1)     CALL 'SYS$ASSIGN' USING BY DESCRIPTOR TERM
38                                  BY REFERENCE TTCHAN
39                                  BY VALUE 0 0
40                            GIVING STAT.
41          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
42      *
43      *   Enable control-C AST
44          COMPUTE FUNC = IO$_SETMODE + IO$M_CTRLCAST
45          CALL 'SYS$QIOW' USING BY VALUE 1 TTCHAN FUNC
46                                BY REFERENCE STATUS_BLOCK
47  (2)                           BY VALUE 0 0 AST-PROC
48                                BY REFERENCE TTCHAN
49                                BY VALUE 0 0 0 0
50                        GIVING STAT.
51          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
52          IF IOSB-STATUS IS FAILURE CALL 'LIB$STOP'
53                        USING BY VALUE IOSB-STATUS.
54      *
55      *   Issue timed read, without echoing chars, giving a prompt
56          COMPUTE FUNC = IO$_READPROMPT + IO$M_TIMED + IO$M_NOECHO
57          CALL 'SYS$QIOW' USING BY VALUE 2 TTCHAN FUNC
58                                BY REFERENCE STATUS-BLOCK
59                                BY VALUE 0 0
60  (3)                           BY REFERENCE BUFFER
61                                BY VALUE 10 5 0
62                                BY REFERENCE PROMPT
63                                BY VALUE 48
64                        GIVING STAT.
65          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
66          DISPLAY ' '.
```

Example 3 Using $QIO to Specify I/O Operations and Respond to CTRL/Cs (Sheet 1 of 2)

```
       *
       *   Check status of I/O and issue appropriate message
           EVALUATE IOSB-STATUS
5            WHEN SS$_NORMAL DISPLAY 'You typed: ' BUFFER(1:IOSB-COUNT)
4          { WHEN SS$_CONTROLC DISPLAY 'Request was cancelled'
           { WHEN SS$_TIMEOUT DISPLAY 'You did not respond in time'
           END-EVALUATE.
           STOP RUN.
       END PROGRAM CONTROLC.


       IDENTIFICATION DIVISION.
       PROGRAM-ID. AST-PROC.
       *
       *   Control C AST Routine
       *
       DATA DIVISION.
       WORKING-STORAGE SECTION.

       01  STAT            PIC S9(9)   COMP.
       LINKAGE SECTION.

       01  TTCHAN  PIC S9(4)  COMP.

       PROCEDURE DIVISION USING TTCHAN.
       BEGIN.
       *
       *   Cancel all I/Os on terminal channel
6          CALL 'SYS$CANCEL' USING BY VALUE TTCHAN GIVING STAT.
           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
       *
       *   Tell user control-c AST routine was entered
7          DISPLAY 'YOU TYPED A CONTROL-C'.
           EXIT PROGRAM.
       END PROGRAM AST-PROC.
```

```
$ COBOL CONTROLC
$ LINK CONTROLC
$ RUN CONTROLC
You have 5 seconds to enter data, or type ctrl-C
*EXIT*

YOU TYPED A CONTROL-C

Request was cancelled
$


$ RUN CONTROLC
You have 5 seconds to enter data, or type ctrl-C

You typed: NANCY
$


$ RUN CONTROLC
You have 5 seconds to enter data, or type ctrl-C

You did not respond in time
$
$
```

Example 3 Using $QIO to Specify I/O Operations and Respond to CTRL/Cs (Sheet 2 of 2)

Example 4

This example shows how to use SMG$ procedures to handle unsolicited input from one or more terminals.

❶ The program prompts for the number of terminals to set up for unsolicited input.

❷ The program prompts for the name of a given terminal to set up for unsolicited input. When you run this program, always include a colon when specifying a terminal name other than TT (for example, TTE7:). However, when specifying your own terminal, the name TT can be used without a colon.

❸ The program uses SMG$ procedures to output a menu to the given terminal. A pasteboard, virtual displays, and a virtual keyboard are created for each terminal. The identification numbers for the pasteboard, virtual displays, and virtual keyboards are stored in separate arrays such that the first element in each array goes with the first terminal set up, the second element with the second terminal and so on.

❹ SMG$ENABLE_UNSOLICITED_INPUT is used to establish an AST procedure that is invoked when unsolicited input is received on the given terminal. SMG$ENABLE_UNSOLICITED_INPUT is passed the pasteboard identification number, the name of the AST procedure to be invoked upon receiving unsolicited input, and the index into the arrays containing the identification numbers for the given terminal.

❺ When the AST procedure is invoked, it is passed the pasteboard identification number and the index into the arrays for the terminal that has re ceived unsolicited input.

❻ SMG$READ_STRING is used to read the unsolicited input from the terminal.

❼ The program responds according to the menu selection received from the terminal.

No sample run is included for this example because the program requires a video terminal to execute properly.

```
1       *                                                    UNSOLICIT.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. UNSOLICIT.
5       *
6       *       This program detects and monitors unsolicited input
7       *       from 1 or more terminal(s) where no one is logged in.
8       *       The program promts for the device name(s) of the
9       *       terminal(s) to be monitored.
10      *
11      DATA DIVISION.
12      WORKING-STORAGE SECTION.
13      01  EX_ARRAY EXTERNAL.
14          02  MENUDSP_ID  PIC S9(9) COMP OCCURS 4 TIMES.
15          02  INDSP_ID    PIC S9(9) COMP OCCURS 4 TIMES.
16          02  OUTDSP_ID   PIC S9(9) COMP OCCURS 4 TIMES.
17          02  KB_ID       PIC S9(9) COMP OCCURS 4 TIMES.
18          02  I_ARRY      PIC S9(9) COMP OCCURS 4 TIMES.
19      01  IN_ARRAY.
20          02  TERM_NAME   PIC X(5) OCCURS 4 TIMES.
21          02  PB_ID       PIC S9(9) COMP OCCURS 4 TIMES.
22      01  ASTRTN          PIC S9(9) COMP VALUE EXTERNAL ASTRTN.
23      01  IN_NUM          PIC S9.
24      01  CT              PIC S9(9) COMP.
25      01  NUM_TERMS       PIC S9(9) COMP.
26      01  I               PIC S9(9) COMP.
27      01  STAT            PIC S9(9) COMP.
28      01  ROW             PIC S9(9) COMP.
29      01  COL             PIC S9(9) COMP.
30      01  D1_ROWS         PIC S9(9) COMP VALUE 9.
31      01  D1_COLS         PIC S9(9) COMP VALUE 25.
32      01  D2_ROWS         PIC S9(9) COMP VALUE 2.
33      01  D2_COLS         PIC S9(9) COMP VALUE 40.
34      01  D3_ROWS         PIC S9(9) COMP VALUE 6.
35      01  D3_COLS         PIC S9(9) COMP VALUE 70.
36
37      PROCEDURE DIVISION.
38      BEGIN.
39      *   Prompt for number of terminals
40  (1)     DISPLAY 'Enter the number of unallocated terminals'.
41          ACCEPT IN_NUM.
42          MOVE IN_NUM TO NUM_TERMS.
43
```

Example 4 SMG$ Procedures to Handle Unsolicited Input (Sheet 1 of 6)

```
44        *   perform the operations in the loop once for each
45        *   terminal to be monitored
46            PERFORM VARYING CT FROM 0 BY 1 UNTIL CT = NUM_TERMS
47        *
48               COMPUTE I = CT + 1
49               MOVE I TO I_ARRY(I)
50
51        *      prompt for remote terminal name
52     2          DISPLAY 'Enter terminal name'
53               ACCEPT TERM_NAME(I)
54        *
55        *      establish a pasteboard, to be written to the remote
56        *      terminal
57               CALL 'SMG$CREATE_PASTEBOARD' USING
58                           BY REFERENCE PB_ID(I)
59                           BY DESCRIPTOR TERM_NAME(I)
60                           BY VALUE 0 0 GIVING STAT
61               IF STAT IS FAILURE THEN
62                    CALL 'LIB$STOP' USING BY VALUE STAT
63               END-IF
64        *
65        *      establish the unallocated terminal as virtual keyboard.
66               CALL 'SMG$CREATE_VIRTUAL_KEYBOARD' USING
67                           BY REFERENCE KB_ID(I)
68                           BY DESCRIPTOR TERM_NAME(I)
69                           BY VALUE 0 0 GIVING STAT
70               IF STAT IS FAILURE THEN
71                    CALL 'LIB$STOP' USING BY VALUE STAT
72               END-IF
73     3  *
74        *      establish a virtual display for displaying the menu.
75               CALL 'SMG$CREATE_VIRTUAL_DISPLAY' USING
76                           BY REFERENCE D1_ROWS D1_COLS MENUDSP_ID(I)
77                           BY VALUE 0 0 0 GIVING STAT
78               IF STAT IS FAILURE THEN
79                    CALL 'LIB$STOP' USING BY VALUE STAT
80               END-IF
81        *
82        *      establish a virtual display for setting input.
83               CALL 'SMG$CREATE_VIRTUAL_DISPLAY' USING
84                           BY REFERENCE D2_ROWS D2_COLS INDSP_ID(I)
85                           BY VALUE 0 0 0 GIVING STAT
86               IF STAT IS FAILURE THEN
87                    CALL 'LIB$STOP' USING BY VALUE STAT
88               END-IF
89        *
```

Example 4 SMG$ Procedures to Handle Unsolicited Input (Sheet 2 of 6)

```
90      *       establish a virtual display for printing messages.
91              CALL 'SMG$CREATE_VIRTUAL_DISPLAY' USING
92                           BY REFERENCE D3_ROWS D3_COLS OUTDSP_ID(I)
93                           BY VALUE 0 0 0 GIVING STAT
94              IF STAT IS FAILURE THEN
95                   CALL 'LIB$STOP' USING BY VALUE STAT
96              END-IF
97      *
98      *       paste the virtual display to the screen, starting at
99      *       row 2, column 5
100             MOVE 2 TO ROW
101             MOVE 5 TO COL
102             CALL 'SMG$PASTE_VIRTUAL_DISPLAY' USING
103                          BY REFERENCE MENUDSP_ID(I) PB_ID(I) ROW COL
104                          GIVING STAT
105             IF STAT IS FAILURE THEN
106                  CALL 'LIB$STOP' USING BY VALUE STAT
107             END-IF
108     *
109     *       paste the input virtual display to the screen, starting at
110     *       row 11, column 5
111             MOVE 11 TO ROW
112             MOVE 5 TO COL
113             CALL 'SMG$PASTE_VIRTUAL_DISPLAY' USING
114                          BY REFERENCE INDSP_ID(I) PB_ID(I) ROW COL
115                          GIVING STAT
116     (3)     IF STAT IS FAILURE THEN
117             CALL 'LIB$STOP' USING BY VALUE STAT
118             END-IF
119     *
120     *       paste the output virtual display to the screen, starting at
121     *       row 14,column 5
122             MOVE 14 TO ROW
123             MOVE 5 TO COL
124             CALL 'SMG$PASTE_VIRTUAL_DISPLAY' USING
125                          BY REFERENCE OUTDSP_ID(I) PB_ID(I) ROW COL
126                          GIVING STAT
127             IF STAT IS FAILURE THEN
128                  CALL 'LIB$STOP' USING BY VALUE STAT
129             END-IF
130     *
131     *       display the menu
132             MOVE 2 TO ROW
133             CALL 'SMG$PUT_LINE' USING
134                          BY REFERENCE MENUDSP_ID(I)
135                          BY DESCRIPTOR 'MAIN MENU'
136                          BY REFERENCE ROW
137                          BY VALUE 0 0 0 0 GIVING STAT
138             IF STAT IS FAILURE THEN
139                  CALL 'LIB$STOP' USING BY VALUE STAT
140             END-IF
```

Example 4 SMG$ Procedures to Handle Unsolicited Input (Sheet 3 of 6)

```
141  ❸          CALL 'SMG$PUT_LINE' USING
142                        BY REFERENCE MENUDSP_ID(I)
143                        BY DESCRIPTOR 'H - HELP'
144                        BY VALUE 0 0 0 0 0 GIVING STAT
145             IF STAT IS FAILURE THEN
146                  CALL 'LIB$STOP' USING BY VALUE STAT
147             END-IF
148             CALL 'SMG$PUT_LINE' USING
149                        BY REFERENCE MENUDSP_ID(I)
150                        BY DESCRIPTOR 'C - CREATE'
151                        BY VALUE 0 0 0 0 0 GIVING STAT
152             IF STAT IS FAILURE THEN
153                  CALL 'LIB$STOP' USING BY VALUE STAT
154             END-IF
155             CALL 'SMG$PUT_LINE' USING
156                        BY REFERENCE MENUDSP_ID(I)
157                        BY DESCRIPTOR 'E - EXIT'
158                        BY REFERENCE ROW
159                        BY VALUE 0 0 0 0 GIVING STAT
160             IF STAT IS FAILURE THEN
161                  CALL 'LIB$STOP' USING BY VALUE STAT
162             END-IF
163             CALL 'SMG$PUT_LINE' USING
164                        BY REFERENCE MENUDSP_ID(I)
165                        BY DESCRIPTOR 'ENTER CAPITAL H, C, or E'
166                        BY VALUE 0 0 0 0 0 GIVING STAT
167             IF STAT IS FAILURE THEN
168                  CALL 'LIB$STOP' USING BY VALUE STAT
169             END-IF
170        *
171        *    enable unsolicited input from the remote terminal
172             CALL 'SMG$ENABLE_UNSOLICITED_INPUT' USING
173                        BY REFERENCE PB_ID(I)
174  ❹                     BY VALUE ASTRTN
175                        BY REFERENCE I_ARRY(I) GIVING STAT
176             IF STAT IS FAILURE THEN
177                  CALL 'LIB$STOP' USING BY VALUE STAT
178             END-IF
179         END-PERFORM.
180        *
181        *    hibernate until awoken by AST routine.
182             CALL 'SYS$HIBER' GIVING STAT.
183             IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
184        *
185             END PROGRAM UNSOLICIT.
186
187
```

Example 4 SMG$ Procedures to Handle Unsolicited Input (Sheet 4 of 6)

```
188       IDENTIFICATION DIVISION.
189       PROGRAM-ID. ASTRTN.
190       *
191       *   This AST routine is invoked when unsolicited input is
192       *   entered to the terminal.
193       *
194       DATA DIVISION.
195       WORKING-STORAGE SECTION.
196
197       01  EX_ARRAY EXTERNAL.
198           02  MENUDSP_ID  PIC S9(9) COMP OCCURS 4 TIMES.
199           02  INDSP_ID    PIC S9(9) COMP OCCURS 4 TIMES.
200           02  OUTDSP_ID   PIC S9(9) COMP OCCURS 4 TIMES.
201           02  KB_ID       PIC S9(9) COMP OCCURS 4 TIMES.
202           02  I_ARRY      PIC S9(9) COMP OCCURS 4 TIMES.
203       01  STAT            PIC S9(9) COMP.
204       01  OUT_STR         PIC X.
205       01  PROMT1          PIC X(17) VALUE 'Enter your choice'.
206       01  PROMT2     PIC X(31) VALUE 'Enter C to show your creativity'.
207       01  PROMT3     PIC X(40) VALUE 'You entered creativity mode. You may'.
208       01  PROMT4     PIC X(40) VALUE 'sing or dance. Enter H or E to go on'.
209       01  PROMT5     PIC X(30) VALUE 'ERROR please enter H or C or E'.
210       *
211       LINKAGE SECTION.
212    5  01  PB_ID      PIC S9(9) COMP.
213       01  I          PIC S9(9) COMP.
214
215       PROCEDURE DIVISION USING PB_ID I.
216       BEGIN.
217       *   prompt user and accept input
218           CALL 'SMG$ERASE_DISPLAY' USING
219                   BY REFERENCE INDSP_ID(I)
220                   BY VALUE 0 0 0 0 GIVING STAT.
221           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
222    6      CALL 'SMG$READ_STRING' USING
223                   BY REFERENCE KB_ID(I)
224                   BY DESCRIPTOR OUT_STR
225                   BY VALUE 0 0 0 0 0 0 0
226                   BY REFERENCE INDSP_ID(I) GIVING STAT.
227           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
228
229       *   act on selection...
230       *   if user selected 'H', display the help message
231    7      IF OUT_STR = 'H' THEN
232                 CALL 'SMG$ERASE_DISPLAY' USING
233                           BY REFERENCE OUTDSP_ID(I)
234                           BY VALUE 0 0 0 0 GIVING STAT
235                 IF STAT IS FAILURE THEN
236                           CALL 'LIB$STOP' USING BY VALUE STAT
237                 END-IF
238                 CALL 'SMG$PUT_LINE' USING
239                           BY REFERENCE OUTDSP_ID(I)
240                           BY DESCRIPTOR PROMT2
241                           BY VALUE 0 0 0 0 0 GIVING STAT
242                 IF STAT IS FAILURE THEN
243                           CALL 'LIB$STOP' USING BY VALUE STAT
244                 END-IF
245           ELSE
246
```

Example 4 SMG$ Procedures to Handle Unsolicited Input (Sheet 5 of 6)

```
247       *    if user selected 'C', enter creativity mode
248            IF OUT_STR = 'C' THEN
249                  CALL 'SMG$ERASE_DISPLAY' USING
250                            BY REFERENCE OUTDSP_ID(I)
251                            BY VALUE 0 0 0 0 GIVING STAT
252                  IF STAT IS FAILURE THEN
253                            CALL 'LIB$STOP' USING BY VALUE STAT
254                  END-IF
255                  CALL 'SMG$PUT_LINE' USING
256                            BY REFERENCE OUTDSP_ID(I)
257                            BY DESCRIPTOR PROMT3
258                            BY VALUE 0 0 0 0 0 GIVING STAT
259                  IF STAT IS FAILURE THEN
260                            CALL 'LIB$STOP' USING BY VALUE STAT
261                  END-IF
262                  CALL 'SMG$PUT_LINE' USING
263                            BY REFERENCE OUTDSP_ID(I)
264                            BY DESCRIPTOR PROMT4
265                            BY VALUE 0 0 0 0 0 GIVING STAT
266                  IF STAT IS FAILURE THEN
267                            CALL 'LIB$STOP' USING BY VALUE STAT
268                  END-IF
269            ELSE
270
271       *    if user selected 'E', wake the main program and exit
272            IF OUT_STR = 'E' THEN
273                  CALL 'SMG$ERASE_DISPLAY' USING
274                            BY REFERENCE OUTDSP_ID(I)
275                            BY VALUE 0 0 0 0 GIVING STAT
276                  IF STAT IS FAILURE THEN
277                            CALL 'LIB$STOP' USING BY VALUE STAT
278                  END-IF
279                  CALL 'SMG$PUT_LINE' USING
280                            BY REFERENCE OUTDSP_ID(I)
281                            BY DESCRIPTOR 'Exiting Program'
282                            BY VALUE 0 0 0 0 0 GIVING STAT
283                  IF STAT IS FAILURE THEN
284                            CALL 'LIB$STOP' USING BY VALUE STAT
285                  END-IF
286                  CALL 'SYS$WAKE' USING
287                            BY VALUE 0 0 GIVING STAT
288                  IF STAT IS FAILURE THEN
289                            CALL 'LIB$STOP' USING BY VALUE STAT
290                  END-IF
291            ELSE
292
293       *    if selection invalid, display an error message
294                  CALL 'SMG$ERASE_DISPLAY' USING
295                            BY REFERENCE OUTDSP_ID(I)
296                            BY VALUE 0 0 0 0 GIVING STAT
297                  IF STAT IS FAILURE THEN
298                            CALL 'LIB$STOP' USING BY VALUE STAT
299                  END-IF
300                  CALL 'SMG$PUT_LINE' USING
301                            BY REFERENCE OUTDSP_ID(I)
302                            BY DESCRIPTOR PROMT5
303                            BY VALUE 0 0 0 0 0 GIVING STAT
304                  IF STAT IS FAILURE THEN
305                            CALL 'LIB$STOP' USING BY VALUE STAT
306                  END-IF
307            END-IF
308            END-IF
309            END-IF.
310       END PROGRAM ASTRTN.
$
```

Example 4 SMG$ Procedures to Handle Unsolicited Input (Sheet 6 of 6)

Example 5

This example shows how to send escape and control sequences directly to VT100/VT200 class terminals from a program.

❶ The SPECIAL NAMES paragraph defines the ASCII characters to be used in the escape sequences.

❷ The escape sequences must be stored as consecutive bytes.

❸ Writing this sequence of bytes will turn off all special video attributes, and then turn on blinking and underline. All text displayed from this point on will assume those attributes.

❹ Writing this escape sequence will cause text on that line to be double-width. This command affects only a single line.

❺ Before exiting, terminal characteristics are reset to the characteristics assigned to the terminal when it was turned on.

No sample run is included for this example because the program requires a video terminal to execute properly.

```
 1       *                                                USEESCAPE.COB
 2       IDENTIFICATION DIVISION.
 3       *
 4       PROGRAM-ID. USEESCAPE.
 5       *
 6       *   This program directly controls the cursor using escape
 7       *   sequences for VT100 class terminals.
 8       *
 9       ENVIRONMENT DIVISION.
10       CONFIGURATION SECTION.
11       SPECIAL-NAMES.
12          {SYMBOLIC CHARACTERS ESCAPE  P36 P49 P53 P54 P55 P60
13    (1)   {                            P92 P100 P110
14          {                ARE   28 36 49 53 54 55 60 92 100 110.
15       DATA DIVISION.
16       WORKING-STORAGE SECTION.
17
18       01    CLEAR-ATTRIB.
19             03  FILLER         PIC X    VALUE ESCAPE.
20             03  FILLER         PIC X    VALUE P92.
21             03  FILLER         PIC X    VALUE P49.
22             03  FILLER         PIC X    VALUE P60.
23             03  FILLER         PIC X    VALUE P53.
24             03  FILLER         PIC X    VALUE P60.
25             03  FILLER         PIC X    VALUE P54.
26    (2)      03  FILLER         PIC X    VALUE P110.
27       01    SET-ATTRIB.
28             03  FILLER     PIC X    VALUE ESCAPE.
29             03  FILLER         PIC X    VALUE P36.
30             03  FILLER         PIC X    VALUE P55.
31       01    RESET-ATTRIB.
32             03  FILLER         PIC X    VALUE ESCAPE.
33             03  FILLER         PIC X    VALUE P100.
34       01    TEXT-DISP          PIC X(1).
35       01    HI             PIC X(2) VALUE 'HI'.
36
37       PROCEDURE DIVISION.
38       BEGIN.
39       *
40       *   Set terminal characteristics with escape sequences
41           DISPLAY CLEAR-ATTRIB HI.
42       *
43    (3)*   Show characteristics by writing text
44           DISPLAY 'Please type RETURN to continue' WITH NO ADVANCING.
45               ACCEPT TEXT-DISP.
46       *
47       *   Loop twice with double-width text
48           PERFORM 2 TIMES
49    (4)            DISPLAY SET-ATTRIB WITH NO ADVANCING
50                   DISPLAY 'Please type RETURN to continue'
51                   ACCEPT TEXT-DISP
52           END-PERFORM.
53       *
54       *   Reset original (power-up) characteristics
55    (5)    DISPLAY RESET-ATTRIB.
56           STOP RUN.
*
```

Example 5 Using Escape Sequences to Control the Cursor Directly

Lab Exercises

Before you begin Exercises 1 and 2, read the information about the RTL SMG$ facility in the *VAX/VMS Run-Time Library Routines Reference Manual*.

1. Write a program using RTL SMG$ facility to do the following:

   a. Display an underlined message on the screen.

   b. Prompt the user to enter three data items. After the user enters the data items, the screen should be formatted as shown below. The user's input is represented by xxxxxx.

```
------5-----10-----15-----20-----25-----30-----35-----40-----45-----50-----55-----60-----65
:                                                                                          : 01
: Personal Information                                                                     : 02
: ----------------------                                                                   : 03
:                                                                                          : 04
: Your Name: xxxxxxxxxxxxxxxxx                                                             : 05
:                                                                                          : 06
: Your Cat's Name: xxxxxxxxxxxxxxxxxx                                                      : 07
:                                                                                          : 08
: Your Dog's Name: xxxxxxxxxxxxxxxxx                                                       : 09
:                                                                                          : 10
:                                                                                          : 11
:                                                                                          : 12
:                                                                                          : 13
:                                                                                          : 14
:                                                                                          : 15
:                                                                                          : 16
:                                                                                          : 17
:                                                                                          : 18
:                                                                                          : 19
:                                                                                          : 20
:                                                                                          : 21
-------------------------------------------------------------------------------------------
```

c. After accepting the user's input, draw a line on the screen, display another underlined message, and display the three data items input by the user. The screen should be formatted as shown below.

```
------5-----10-----15-----20-----25-----30-----35-----40-----45-----50-----55-----60-----65
:                                                                                    : 01
:  Personal Information                                                              : 02
:   ------------------------                                                         : 03
:                                                                                    : 04
:  Your Name: xxxxxxxxxxxxxxxxx                                                      : 05
:                                                                                    : 06
:  Your Cat's Name: xxxxxxxxxxxxxxxxxx                                               : 07
:                                                                                    : 08
:  Your Dog's Name: xxxxxxxxxxxxxxxxx                                                : 09
:                                                                                    : 10
:-------------------------------------------------                                   : 11
:                                                                                    : 12
:  Personal Report                                                                   : 13
:  -------------------                                                              : 14
:                                                                                    : 15
:  Your Name is xxxxxxxxxxxxxxxx                                                     : 16
:                                                                                    : 17
:  Your Cat's Name is xxxxxxxxxxxxxxxxxxx                                            : 18
:                                                                                    : 19
:  Your Dog's Name is  xxxxxxxxxxxxxxxxx                                             : 20
:                                                                                    : 21
-------------------------------------------------------------------------------------
```

2. Modify the program you wrote for Exercise 1 to do the following:

   a. Put an unlabeled border around the entire display.

   b. After accepting and displaying the user's input, move the entire bordered display to another portion of the screen.

      The portion of the screen where the display originated will become blank and another portion of the screen will be filled with the display. After moving the display, the screen will appear as follows:

```
------5-----10-----15-----20-----25-----30-----35-----40-----45-----50-----55-----60-----65
:                              :-------------------------------------------------:  :  01
:                              :  Personal Information                           :  :  02
:                              :  --------------------                           :  :  03
:                              :                                                 :  :  04
:                              :  Your Name: xxxxxxxxxxxxxxxxx                   :  :  05
:                              :                                                 :  :  06
:                              :  Your Cat's Name: xxxxxxxxxxxxxxxxx             :  :  07
:                              :                                                 :  :  08
:                              :  Your Dog's Name:  xxxxxxxxxxxxxxxx             :  :  09
:                              :                                                 :  :  10
:                              :-------------------------------------------------:  :  11
:                              :                                                 :  :  12
:                              :  Personal Report                                :  :  13
:                              :  ---------------                                :  :  14
:                              :                                                 :  :  15
:                              :  Your Name is xxxxxxxxxxxxxxxxxxx               :  :  16
:                              :                                                 :  :  17
:                              :  Your Cat's Name is xxxxxxxxxxxxxxxx            :  :  18
:                              :                                                 :  :  19
:                              :  Your Dog's Name is xxxxxxxxxxxxxxx             :  :  20
:                              :-------------------------------------------------:  :  21
-------------------------------------------------------------------------------------
```

3. The program LAB3.COB does the following:
   a. Establishes an AST routine that responds to CTRL/A characters
   b. Issues a timed read to the terminal (waiting for CTRL/A)
   c. If a CTRL/A is typed, does the following:
      1) Cancels the timed read
      2) Reads a three-character code from the terminal
      3) Displays the code to the user with a separate $QIOW
      4) Reissues the timed read
   d. If a timeout occurs, exits the program
   e. If the timed read completes successfully, reissues another timed read
   f. Modify LAB3.COB of this module so that if CTRL/B is typed it:
      1) Cancels the timed read
      2) Reads data from the terminal, converting any lowercase characters to uppercase
      3) Displays the data typed to the user
      4) Reissues the timed read

Provided File

```
*                                                                LAB3.COB
IDENTIFICATION DIVISION.
*
PROGRAM-ID.     LAB3.
*
*     This program illustrates the use of the out of band
*     scheme to respond to special control characters.
*
DATA DIVISION.
WORKING-STORAGE SECTION.

01    AST1              PIC  S9(9)   COMP    VALUE  EXTERNAL  AST1.
01    STATUS-BLOCK.
      03  IOSB-STAT     PIC  S9(4)   COMP.
      03  IOSB-COUNT    PIC  S9(4)   COMP.
      03  IOSB-INFO     PIC  S9(9)   COMP.
01    DUMMY             PIC  S9(9)   COMP.
01    AMASK                          COMP.
      03       AMASK1   PIC  S9(9)   COMP    VALUE  0.
      03       AMASK2   PIC  S9(9)   COMP    VALUE  2.
01    FUNC              PIC  S9(9)   COMP.
01    IO$_SETMODE       PIC  S9(9)   COMP    VALUE  EXTERNAL  IO$_SETMODE.
01    IO$M_OUTBAND      PIC  S9(9)   COMP    VALUE  EXTERNAL  IO$M_OUTBAND.
01    IO$_READPROMPT    PIC  S9(9)   COMP    VALUE  EXTERNAL  IO$_READPROMPT.
01    IO$M_TIMED        PIC  S9(9)   COMP    VALUE  EXTERNAL  IO$M_TIMED.
01    SS$_TIMEOUT       PIC  S9(9)   COMP    VALUE  EXTERNAL  SS$_TIMEOUT.
01    SS$_ABORT         PIC  S9(9)   COMP    VALUE  EXTERNAL  SS$_ABORT.
01    SS$_NORMAL        PIC  S9(9)   COMP    VALUE  EXTERNAL  SS$_NORMAL.
01    TT                PIC  X(11)           VALUE  'SYS$COMMAND'.
01    STAT              PIC  S9(9)   COMP.
01    CHANNEL                        COMP.
      03       TTCHAN                OCCURS 2 TIMES  PIC S9(4) COMP.
01    PROMPT            PIC  X(44)   VALUE
      'You have 10 seconds to type CTRL/A'.
```

(Sheet 1 of 3)

```
PROCEDURE DIVISION.
BEGIN.
*   Assign channels to terminal
    CALL 'SYS$ASSIGN' USING BY DESCRIPTOR TT
                            BY REFERENCE TTCHAN(1)
                            BY VALUE 0 0
                            GIVING STAT.
    IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
    CALL 'SYS$ASSIGN' USING BY DESCRIPTOR TT
                            BY REFERENCE TTCHAN(2)
                            BY VALUE 0 0
                            GIVING STAT.
    IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
    CALL 'LIB$PUT_COMMON' USING BY DESCRIPTOR CHANNEL.
*
*   Establish CTRL/A AST routine
    COMPUTE FUNC = IO$_SETMODE + IO$M_OUTBAND
    CALL 'SYS$QIOW' USING BY VALUE 1 TTCHAN(2) FUNC
                                0 0 0 AST1
                            BY REFERENCE AMASK
                            BY VALUE 0 0 0 0
                          GIVING STAT.
    IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
*
*   Wait for user to enter data
10. COMPUTE FUNC = IO$_READPROMPT + IO$M_TIMED
    CALL 'SYS$QIOW' USING   BY VALUE 3 TTCHAN(1) FUNC
                            BY REFERENCE STATUS-BLOCK
                            BY VALUE 0 0
                            BY REFERENCE DUMMY
                            BY VALUE 80 10 0
                            BY REFERENCE PROMPT
                            BY VALUE 44
                          GIVING STAT.
    IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
*
*   Test for abort and timeout codes
    EVALUATE IOSB-STAT
              WHEN SS$_ABORT GO TO 10
              WHEN SS$_NORMAL GO TO 10
              WHEN SS$_TIMEOUT DISPLAY 'Program timed out'
              WHEN OTHER DISPLAY 'Unknown reason for program failure'
    END-EVALUATE.
END PROGRAM LAB3.
```

(Sheet 2 of 3)

```
IDENTIFICATION DIVISION.
*
PROGRAM-ID.       AST1.
*
*    CTRL/A AST Routine
*
DATA DIVISION.
WORKING-STORAGE SECTION.

01   AMES             PIC    X(31)          VALUE
                      ' Please enter 3 character code'.
01   FUNC             PIC    S9(9)   COMP.
01   IO$_READPROMPT   PIC    S9(9)   COMP   VALUE   EXTERNAL   IO$_READPROMPT.
01   IO$M_NOECHO      PIC    S9(9)   COMP   VALUE   EXTERNAL   IO$M_NOECHO.
01   BUFFER           PIC    X(10)
01   CHANNEL          COMP.
     03      TTCHAN   OCCURS 2 TIMES         PIC    S9(4)  COMP.
01   CHAN1            PIC    S9(9)   COMP.
01   STATUS-BLOCK.
     03  IOSB-STAT    PIC    S9(4)   COMP.
     03  IOSB-COUNT   PIC    S9(4)   COMP.
     03  IOSB-INFO    PIC    S9(9)   COMP.
01   STAT             PIC    S9(9)   COMP.

PROCEDURE DIVISION.
BEGIN.
     CALL 'LIB$GET_COMMON'USING BY DESCRIPTOR CHANNEL.
     MOVE TTCHAN(1) TO CHAN1.
     CALL 'SYS$CANCEL' USING BY VALUE CHAN1.
     DISPLAY ''.
     COMPUTE FUNC = IO$_READPROMPT + IO$M_NOECHO
     CALL 'SYS$QIOW' USING BY VALUE 1 TTCHAN(2) FUNC
                           BY REFERENCE STATUS-BLOCK
                           BY VALUE 0 0
                           BY REFERENCE BUFFER
                           BY VALUE 3 0 0
                           BY REFERENCE AMES
                           BY VALUE 31
                      GIVING STAT.
     IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
     IF IOSB-STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE IOSB-STAT.
     DISPLAY 'You typed: ' BUFFER(1:IOSB-COUNT).
     EXIT PROGRAM.
END PROGRAM AST1.
```

(Sheet 3 of 3)

Lab Solutions

1.

```
1       *                                                    LABSOL1.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL1.
5       *
6       *   This program calls Run-Time Library Screen Management
7       *   routines to format screen output, accept user input, and
8       *   display the data entered.
9       *
10      DATA DIVISION.
11      WORKING-STORAGE SECTION.
12      *
13      01 SMG$M_UNDERLINE      PIC S9(9) COMP VALUE 8.
14      01 NAME.
15          02  NAME_LABL       PIC X(13) VALUE 'Your name is '.
16          02  Y_NAME          PIC X(15).
17      01 CAT.
18          02  CAT_LABL        PIC X(20) VALUE 'Your Cat''s name is '.
19          02  Y_CAT           PIC X(15).
20      01 DOG.
21          02  DOG_LABL        PIC X(20) VALUE 'Your Dog''s name is '.
22          02  Y_DOG           PIC X(15).
23      01 PROMT1               PIC X(11) VALUE 'Your Name: '.
24      01 PROMT2               PIC X(17) VALUE 'Your Cat''s Name: '.
25      01 PROMT3               PIC X(17) VALUE 'Your Dog''s Name: '.
26      01 NAME_LEN             PIC S9(9) COMP.
27      01 CAT_LEN              PIC S9(9) COMP.
28      01 DOG_LEN              PIC S9(9) COMP.
29      01 STAT                 PIC S9(9) COMP.
30      01 KEYBOARD_ID          PIC S9(9) COMP.
31      01 DISPLAY_ID           PIC S9(9) COMP.
32      01 NEW_PID              PIC S9(9) COMP.
33      01 N_ROW                PIC S9(9) COMP.
34      01 N_COL                PIC S9(9) COMP.
35      01 END_ROW              PIC S9(9) COMP.
36      01 END_COL              PIC S9(9) COMP.
37      01 DBL_SPC              PIC S9(9) COMP VALUE 2.
38
39      *
40      PROCEDURE DIVISION.
41      BEGIN.
42      *
43      *   establish terminal keyboard as virtual keyboard
44          CALL 'SMG$CREATE_VIRTUAL_KEYBOARD' USING
45                          BY REFERENCE KEYBOARD_ID
46                          BY VALUE 0 0 0 GIVING STAT.
47          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
48      *
49      *   establish terminal screen as a pasteboard
50          CALL 'SMG$CREATE_PASTEBOARD' USING
51                          BY REFERENCE NEW_PID
52                          BY VALUE 0 0 0 GIVING STAT.
53          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
54      *
55      *   establish a virtual display region
56          MOVE 21 TO N_ROW.
57          MOVE 25 TO N_COL.
58          CALL 'SMG$CREATE_VIRTUAL_DISPLAY' USING
59                          BY REFERENCE N_ROW N_COL DISPLAY_ID
60                          BY VALUE 0 0 0 GIVING STAT.
61          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
```

(Sheet 1 of 3)

```
*
*   Paste the virtual display to the screen
    MOVE 2 TO N_ROW.
    MOVE 2 TO N_COL.
    CALL 'SMG$PASTE_VIRTUAL_DISPLAY' USING
                    BY REFERENCE DISPLAY_ID NEW_PID
                    BY REFERENCE N_ROW N_COL GIVING STAT.
    IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
*
*   display the menu selctions
    CALL 'SMG$PUT_LINE' USING
                    BY REFERENCE DISPLAY_ID
                    BY DESCRIPTOR 'Personal Information'
                    BY REFERENCE DBL_SPC SMG$M_UNDERLINE
                    BY VALUE 0 0 0 GIVING STAT.
    IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.

*   Prompt user and accept input
    CALL 'SMG$READ_STRING' USING
                    BY REFERENCE KEYBOARD_ID
                    BY DESCRIPTOR Y_NAME PROMT1
                    BY VALUE 0 0 0 0
                    BY REFERENCE NAME_LEN
                    BY VALUE 0
                    BY REFERENCE DISPLAY_ID GIVING STAT.
    IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
    MOVE 5 TO N_ROW.
    MOVE 1 TO N_COL.
    CALL 'SMG$SET_CURSOR_ABS' USING
                    BY REFERENCE DISPLAY_ID N_ROW N_COL
                    GIVING STAT.
    IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
    CALL 'SMG$READ_STRING' USING
                    BY REFERENCE KEYBOARD_ID
                    BY DESCRIPTOR Y_CAT PROMT2
                    BY VALUE 0 0 0 0
                    BY REFERENCE CAT_LEN
                    BY VALUE 0
                    BY REFERENCE DISPLAY_ID GIVING STAT.
    IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
    MOVE 7 TO N_ROW.
    MOVE 1 TO N_COL.
    CALL 'SMG$SET_CURSOR_ABS' USING
                    BY REFERENCE DISPLAY_ID N_ROW N_COL
                    GIVING STAT.
    IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
    CALL 'SMG$READ_STRING' USING
                    BY REFERENCE KEYBOARD_ID
                    BY DESCRIPTOR Y_DOG PROMT3
                    BY VALUE 0 0 0 0
                    BY REFERENCE DOG_LEN
                    BY VALUE 0
                    BY REFERENCE DISPLAY_ID GIVING STAT.
    IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
```

(Sheet 2 of 3)

```
116       *
117       *   display the users information
118           MOVE 10 TO N_ROW.
119           MOVE 1 TO N_COL.
120           MOVE 10 TO END_ROW.
121           MOVE 25 TO END_COL.
122           CALL 'SMG$DRAW_LINE' USING
123                           BY REFERENCE DISPLAY_ID N_ROW N_COL
124                           BY REFERENCE END_ROW END_COL
125                           BY VALUE 0 0 GIVING STAT.
126           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
127           MOVE 12 TO N_ROW.
128           MOVE 1 TO N_COL.
129           CALL 'SMG$SET_CURSOR_ABS' USING
130                           BY REFERENCE DISPLAY_ID N_ROW N_COL
131                           GIVING STAT.
132           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
133           CALL 'SMG$PUT_LINE' USING
134                           BY REFERENCE DISPLAY_ID
135                           BY DESCRIPTOR 'Personal Report'
136                           BY REFERENCE DBL_SPC SMG$M_UNDERLINE
137                           BY VALUE 0 0 0 GIVING STAT.
138           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
139           IF (NAME_LEN > 0) THEN
140                   CALL 'SMG$PUT_LINE' USING
141                                   BY REFERENCE DISPLAY_ID
142                                   BY DESCRIPTOR NAME
143                                   BY REFERENCE DBL_SPC
144                                   BY VALUE 0 0 0 0 GIVING STAT
145                   IF STAT IS FAILURE THEN
146                           CALL 'LIB$STOP' USING BY VALUE STAT
147                   END-IF
148           END-IF.
149           IF (CAT_LEN > 0) THEN
150                   CALL 'SMG$PUT_LINE' USING
151                                   BY REFERENCE DISPLAY_ID
152                                   BY DESCRIPTOR CAT
153                                   BY REFERENCE DBL_SPC
154                                   BY VALUE 0 0 0 0 GIVING STAT
155                   IF STAT IS FAILURE THEN
156                           CALL 'LIB$STOP' USING BY VALUE STAT
157                   END-IF
158           END-IF.
159           IF (DOG_LEN > 0) THEN
160                   CALL 'SMG$PUT_LINE' USING
161                                   BY REFERENCE DISPLAY_ID
162                                   BY DESCRIPTOR DOG
163                                   BY REFERENCE DBL_SPC
164                                   BY VALUE 0 0 0 0 GIVING STAT
165                   IF STAT IS FAILURE THEN
166                           CALL 'LIB$STOP' USING BY VALUE STAT
167                   END-IF
168           END-IF.
169           STOP RUN.
```

(Sheet 3 of 3)

2.

```
1       *                                                    LABSOL2.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL2.
5       *
6       *   This program calls Run-Time Library Screen Management
7       *   routines to format screen output, accept user input, and
8       *   display the data entered.  It moves the entire display to
9       *   another area of the screen.
10      *
11      DATA DIVISION.
12      WORKING-STORAGE SECTION.
13      *
14      01 SMG$M_UNDERLINE      PIC S9(9) COMP VALUE 8.
15      01 SMG$M_BORDER         PIC S9(9) COMP VALUE 1.
16      01 NAME.
17          02  NAME_LABL       PIC X(13) VALUE 'Your name is '.
18          02  Y_NAME          PIC X(15).
19      01 CAT.
20          02  CAT_LABL        PIC X(20) VALUE 'Your Cat''s name is '.
21          02  Y_CAT           PIC X(15).
22      01 DOG.
23          02  DOG_LABL        PIC X(20) VALUE 'Your Dog''s name is '.
24          02  Y_DOG           PIC X(15).
25      01      PROMT1           PIC X(11) VALUE 'Your Name: '.
26      01      PROMT2           PIC X(17) VALUE 'Your Cat''s Name: '.
27      01      PROMT3           PIC X(17) VALUE 'Your Dog''s Name: '.
28      01 NAME_LEN              PIC S9(9) COMP.
29      01 CAT_LEN               PIC S9(9) COMP.
30      01 DOG_LEN               PIC S9(9) COMP.
31      01 STAT                  PIC S9(9) COMP.
32      01 KEYBOARD_ID           PIC S9(9) COMP.
33      01 DISPLAY_ID            PIC S9(9) COMP.
34      01 NEW_PID               PIC S9(9) COMP.
35      01 N_ROW                 PIC S9(9) COMP.
36      01 N_COL                 PIC S9(9) COMP.
37      01 END_ROW               PIC S9(9) COMP.
38      01 END_COL               PIC S9(9) COMP.
39      01 DBL_SPC               PIC S9(9) COMP VALUE 2.
40
41      *
42      PROCEDURE DIVISION.
43      BEGIN.
44      *
45      *   establish terminal keyboard as virtual keyboard
46          CALL 'SMG$CREATE_VIRTUAL_KEYBOARD' USING
47                          BY REFERENCE KEYBOARD_ID
48                          BY VALUE 0 0 0 GIVING STAT.
49          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
50      *
51      *   establish terminal screen as a pasteboard
52          CALL 'SMG$CREATE_PASTEBOARD' USING
53                          BY REFERENCE NEW_PID
54                          BY VALUE 0 0 0 GIVING STAT.
55          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
56      *
57      *   establish a virtual display region
58          MOVE 21 TO N_ROW.
59          MOVE 25 TO N_COL.
60          CALL 'SMG$CREATE_VIRTUAL_DISPLAY' USING
61                          BY REFERENCE N_ROW N_COL
62                          BY REFERENCE DISPLAY_ID SMG$M_BORDER
63                          BY VALUE 0 0 GIVING STAT.
64          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
```

(Sheet 1 of 3)

```
65      *
66      *   Paste the virtual display to the screen
67          MOVE 2 TO N_ROW.
68          MOVE 2 TO N_COL.
69          CALL 'SMG$PASTE_VIRTUAL_DISPLAY' USING
70                          BY REFERENCE DISPLAY_ID NEW_PID
71                          BY REFERENCE N_ROW N_COL GIVING STAT.
72          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
73      *
74      *   display the menu selctions
75          CALL 'SMG$PUT_LINE' USING
76                          BY REFERENCE DISPLAY_ID
77                          BY DESCRIPTOR 'Personal Information'
78                          BY REFERENCE DBL_SPC SMG$M_UNDERLINE
79                          BY VALUE 0 0 0 GIVING STAT.
80          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
81
82      *   Prompt user and accept input
83          CALL 'SMG$READ_STRING' USING
84                          BY REFERENCE KEYBOARD_ID
85                          BY DESCRIPTOR Y_NAME PROMT1
86                          BY VALUE 0 0 0 0
87                          BY REFERENCE NAME_LEN
88                          BY VALUE 0
89                          BY REFERENCE DISPLAY_ID GIVING STAT.
90          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
91          MOVE 5 TO N_ROW.
92          MOVE 1 TO N_COL.
93          CALL 'SMG$SET_CURSOR_ABS' USING
94                          BY REFERENCE DISPLAY_ID N_ROW N_COL
95                          GIVING STAT.
96          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
97          CALL 'SMG$READ_STRING' USING
98                          BY REFERENCE KEYBOARD_ID
99                          BY DESCRIPTOR Y_CAT PROMT2
100                         BY VALUE 0 0 0 0
101                         BY REFERENCE CAT_LEN
102                         BY VALUE 0
103                         BY REFERENCE DISPLAY_ID GIVING STAT.
104         IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
105         MOVE 7 TO N_ROW.
106         MOVE 1 TO N_COL.
107         CALL 'SMG$SET_CURSOR_ABS' USING
108                         BY REFERENCE DISPLAY_ID N_ROW N_COL
109                         GIVING STAT.
110         IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
111         CALL 'SMG$READ_STRING' USING
112                         BY REFERENCE KEYBOARD_ID
113                         BY DESCRIPTOR Y_DOG PROMT3
114                         BY VALUE 0 0 0 0
115                         BY REFERENCE DOG_LEN
116                         BY VALUE 0
117                         BY REFERENCE DISPLAY_ID GIVING STAT.
118         IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
```

(Sheet 2 of 3)

```
119         *
120         *   display the users information
121             MOVE 10 TO N_ROW.
122             MOVE 1 TO N_COL.
123             MOVE 10 TO END_ROW.
124             MOVE 25 TO END_COL.
125             CALL 'SMG$DRAW_LINE' USING
126                             BY REFERENCE DISPLAY_ID N_ROW N_COL
127                             BY REFERENCE END_ROW END_COL
128                             BY VALUE 0 0 GIVING STAT.
129             IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
130             MOVE 12 TO N_ROW.
131             MOVE 1 TO N_COL.
132             CALL 'SMG$SET_CURSOR_ABS' USING
133                             BY REFERENCE DISPLAY_ID N_ROW N_COL
134                             GIVING STAT.
135             IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
136             CALL 'SMG$PUT_LINE' USING
137                             BY REFERENCE DISPLAY_ID
138                             BY DESCRIPTOR 'Personal Report'
139                             BY REFERENCE DBL_SPC SMG$M_UNDERLINE
140                             BY VALUE 0 0 0 GIVING STAT.
141             IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
142             IF (NAME_LEN > 0) THEN
143                     CALL 'SMG$PUT_LINE' USING
144                                     BY REFERENCE DISPLAY_ID
145                                     BY DESCRIPTOR NAME
146                                     BY REFERENCE DBL_SPC
147                                     BY VALUE 0 0 0 0 GIVING STAT
148                     IF STAT IS FAILURE THEN
149                             CALL 'LIB$STOP' USING BY VALUE STAT
150                     END-IF
151             END-IF.
152             IF (CAT_LEN > 0) THEN
153                     CALL 'SMG$PUT_LINE' USING
154                                     BY REFERENCE DISPLAY_ID
155                                     BY DESCRIPTOR CAT
156                                     BY REFERENCE DBL_SPC
157                                     BY VALUE 0 0 0 0 GIVING STAT
158                     IF STAT IS FAILURE THEN
159                             CALL 'LIB$STOP' USING BY VALUE STAT
160                     END-IF
161             END-IF.
162             IF (DOG_LEN > 0) THEN
163                     CALL 'SMG$PUT_LINE' USING
164                                     BY REFERENCE DISPLAY_ID
165                                     BY DESCRIPTOR DOG
166                                     BY REFERENCE DBL_SPC
167                                     BY VALUE 0 0 0 0 GIVING STAT
168                     IF STAT IS FAILURE THEN
169                             CALL 'LIB$STOP' USING BY VALUE STAT
170                     END-IF
171             END-IF.
172         *
173         *   move the virtual display to a different part of the screen
174             CALL 'SMG$UNPASTE_VIRTUAL_DISPLAY' USING
175                             BY REFERENCE DISPLAY_ID NEW_PID
176                             GIVING STAT.
177             IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
178
179         *   paste the virtual display to the screen
180             MOVE 2 TO N_ROW.
181             MOVE 40 TO N_COL.
182             CALL 'SMG$PASTE_VIRTUAL_DISPLAY' USING
183                             BY REFERENCE DISPLAY_ID NEW_PID
184                             BY REFERENCE N_ROW N_COL GIVING STAT.
185             IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
186         *
187             STOP RUN.
```

(Sheet 3 of 3)

3.

```
1       *                                                LABSOL3.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL3.
5       *
6       *   This program illustrates the use of the out of band
7       *   scheme to respond to special control characters.
8       *
9       DATA DIVISION.
10      WORKING-STORAGE SECTION.
11
12      01  AST1             PIC S9(9)  COMP  VALUE  EXTERNAL AST1.
13      01  AST2             PIC S9(9)  COMP  VALUE  EXTERNAL AST2.
14      01  STATUS-BLOCK.
15          03   IOSB-STAT   PIC  S9(4)  COMP.
16          03   IOSB-COUNT  PIC  S9(4)  COMP.
17          03   IOSB-INFO   PIC  S9(9)  COMP.
18      01  DUMMY            PIC S9(9)  COMP.
19      01  AMASK                       COMP.
20          03      AMASK1    PIC S9(9)  COMP  VALUE  0.
21          03      AMASK2    PIC S9(9)  COMP  VALUE  2.
22      01  BMASK                       COMP.
23          03   BMASK1       PIC S9(9)  COMP  VALUE  0.
24          03   BMASK2       PIC S9(9)  COMP  VALUE  4.
25      01  FUNC             PIC S9(9)  COMP.
26      01  IO$_SETMODE      PIC S9(9)  COMP  VALUE  EXTERNAL IO$_SETMODE.
27      01  IO$M_OUTBAND     PIC S9(9)  COMP  VALUE  EXTERNAL IO$M_OUTBAND.
28      01  IO$_READPROMPT   PIC S9(9)  COMP  VALUE  EXTERNAL IO$_READPROMPT.
29      01  IO$M_TIMED       PIC S9(9)  COMP  VALUE  EXTERNAL IO$M_TIMED.
30      01  SS$_TIMEOUT      PIC S9(9)  COMP  VALUE  EXTERNAL SS$_TIMEOUT.
31      01  SS$_ABORT        PIC S9(9)  COMP  VALUE  EXTERNAL SS$_ABORT.
32      01  SS$_NORMAL       PIC S9(9)  COMP  VALUE  EXTERNAL SS$_NORMAL.
33      01  TT               PIC X(11)        VALUE  'SYS$COMMAND'.
34      01  STAT             PIC S9(9)  COMP.
35      01  CHANNEL          COMP.
36          03      TTCHAN    OCCURS 3 TIMES   PIC S9(4)  COMP.
37      01  PROMPT           PIC X(44)  VALUE
38          'You have 10 seconds to type CTRL/A or CTRL/B'.
39
40      PROCEDURE DIVISION.
41      BEGIN.
42      *   Assign channels to terminal
43          CALL 'SYS$ASSIGN' USING BY DESCRIPTOR TT
44                                  BY REFERENCE TTCHAN(1)
45                                  BY VALUE 0 0
46                                  GIVING STAT.
47          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
48          CALL 'SYS$ASSIGN' USING BY DESCRIPTOR TT
49                                  BY REFERENCE TTCHAN(2)
50                                  BY VALUE 0 0
51                                  GIVING STAT.
52          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
53          CALL 'SYS$ASSIGN' USING BY DESCRIPTOR TT
54                                  BY REFERENCE TTCHAN(3)
55                                  BY VALUE 0 0
56                                  GIVING STAT.
57          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
58          CALL 'LIB$PUT_COMMON' USING BY DESCRIPTOR CHANNEL.
```

(Sheet 1 of 3)

```
59      *
60      *   Establish CTRL/A and CTRL/B AST routines
61          COMPUTE FUNC = IO$_SETMODE + IO$M_OUTBAND
62          CALL 'SYS$QIOW' USING BY VALUE 1 TTCHAN(2) FUNC
63                                          0 0 0 AST1
64                                BY REFERENCE AMASK
65                                BY VALUE 0 0 0 0
66                             GIVING STAT.
67          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
68          CALL 'SYS$QIOW' USING BY VALUE 1 TTCHAN(3) FUNC
69                                          0 0 0 AST2
70                                BY REFERENCE BMASK
71                                BY VALUE 0 0 0 0
72                             GIVING STAT.
73          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
74      *
75      *   Wait for user to enter data
76      10. COMPUTE FUNC = IO$_READPROMPT + IO$M_TIMED
77          CALL 'SYS$QIOW' USING BY VALUE 3 TTCHAN(1) FUNC
78                                BY REFERENCE STATUS-BLOCK
79                                BY VALUE 0 0
80                                BY REFERENCE DUMMY
81                                BY VALUE 80 10 0
82                                BY REFERENCE PROMPT
83                                BY VALUE 44
84                           GIVING STAT.
85          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
86      *
87      *   Test for abort and timeout codes
88          EVALUATE IOSB-STAT
89                   WHEN SS$_ABORT GO TO 10
90                   WHEN SS$_NORMAL GO TO 10
91                   WHEN SS$_TIMEOUT DISPLAY 'Program timed out'
92                   WHEN OTHER DISPLAY 'Unknown reason for program failure'
93          END-EVALUATE.
94      END PROGRAM LABSOL3.
95
96
97      IDENTIFICATION DIVISION.
98      *
99      PROGRAM-ID. AST1.
100     *
101     *   CTRL/A AST Routine
102     *
103     DATA DIVISION.
104     WORKING-STORAGE SECTION.
105
106     01  AMES              PIC X(31)        VALUE
107                           '  Please enter 3 character code'.
108     01  FUNC              PIC S9(9)  COMP.
109     01  IO$_READPROMPT    PIC S9(9)  COMP  VALUE EXTERNAL IO$_READPROMPT.
110     01  IO$M_NOECHO       PIC S9(9)  COMP  VALUE EXTERNAL IO$M_NOECHO.
111     01  BUFFER            PIC X(10).
112     01  CHANNEL           COMP.
113         03      TTCHAN     OCCURS 3 TIMES   PIC S9(4)  COMP.
114     01  CHAN1             PIC S9(9)  COMP.
115     01  STATUS-BLOCK.
116         03   IOSB-STAT    PIC  S9(4)  COMP.
117         03   IOSB-COUNT   PIC  S9(4)  COMP.
118         03   IOSB-INFO    PIC  S9(9)  COMP.
119     01  STAT              PIC  S9(9)  COMP.
120
```

(Sheet 2 of 3)

```
121         PROCEDURE DIVISION.
122         BEGIN.
123             CALL 'LIB$GET_COMMON' USING BY DESCRIPTOR CHANNEL.
124             MOVE TTCHAN(1) TO CHAN1.
125             CALL 'SYS$CANCEL' USING BY VALUE CHAN1.
126             DISPLAY ' '.
127             COMPUTE FUNC = IO$_READPROMPT + IO$M_NOECHO
128             CALL 'SYS$QIOW' USING BY VALUE 1 TTCHAN(2) FUNC
129                                   BY REFERENCE STATUS-BLOCK
130                                   BY VALUE 0 0
131                                   BY REFERENCE BUFFER
132                                   BY VALUE 3 0 0
133                                   BY REFERENCE AMES
134                                   BY VALUE 31
135                             GIVING STAT.
136             IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
137             IF IOSB-STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE IOSB-STAT.
138             DISPLAY 'You typed: ' BUFFER(1:IOSB-COUNT).
139             EXIT PROGRAM.
140         END PROGRAM AST1.
141
142
143         IDENTIFICATION DIVISION.
144         *
145         PROGRAM-ID. AST2.
146         *
147         *   CTRL/B AST Routine
148         *
149         DATA DIVISION.
150         WORKING-STORAGE SECTION.
151
152         01  BMES             PIC X(38)         VALUE
153                              '  Please enter lower case characters: '.
154         01  FUNC             PIC S9(9)  COMP.
155         01  IO$_READPROMPT   PIC S9(9)  COMP  VALUE EXTERNAL IO$_READPROMPT.
156         01  IO$M_CVTLOW      PIC S9(9)  COMP  VALUE EXTERNAL IO$M_CVTLOW.
157         01  BUFFER           PIC X(10).
158         01  CHANNEL          PIC S9(9)  COMP.
159             03      TTCHAN   OCCURS 3 TIMES   PIC S9(4)  COMP.
160         01  CHAN1            PIC S9(9)  COMP.
161         01  STATUS-BLOCK.
162             03   IOSB-STAT   PIC  S9(4)  COMP.
163             03   IOSB-COUNT  PIC  S9(4)  COMP.
164             03   IOSB-INFO   PIC  S9(9)  COMP.
165         01  STAT             PIC  S9(9)  COMP.
166
167         PROCEDURE DIVISION.
168         BEGIN.
169             CALL 'LIB$GET_COMMON' USING BY DESCRIPTOR CHANNEL.
170             MOVE TTCHAN(1) TO CHAN1.
171             CALL 'SYS$CANCEL' USING BY VALUE CHAN1.
172             DISPLAY ' '.
173             COMPUTE FUNC = IO$_READPROMPT + IO$M_CVTLOW
174             CALL 'SYS$QIOW' USING BY VALUE 0 TTCHAN(3) FUNC
175                                   BY REFERENCE STATUS-BLOCK
176                                   BY VALUE 0 0
177                                   BY REFERENCE BUFFER
178                                   BY VALUE 80 0 0
179                                   BY REFERENCE BMES
180                                   BY VALUE 38
181                             GIVING STAT.
182             IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
183             IF IOSB-STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE IOSB-STAT.
184             DISPLAY 'You typed: ' BUFFER(1:IOSB-COUNT).
185             EXIT PROGRAM.
186         END PROGRAM AST2.
```

(Sheet 3 of 3)

Building Human Interfaces

File Locations

On-line files are provided to help you master this module. The files are located in the directory:

```
DISK$COURSE:[COURSE.V4PROG.COB.INTR]
```

For your convenience, your system manager may have created the following logical name equivalence:

```
DISK$COURSE:[COURSE.V4PROG.COB.INTR] = V4PROG$COB$INTR
```

Two types of files are provided:

1. The Module Programming Examples, which are also listed in this language workbook.
2. Solutions to the Laboratory Exercises.

If you wish to modify and/or compile, link, and run any of the programs, first copy the file into your directory.

Example 1

Example 1 illustrates how to obtain text from a help library. After the initial help request has been satisfied, the user is prompted and can request additional information.

❶ The address of an output routine is a required argument. When requesting the prompting mode (the default mode), an input routine must be specified.

❷ To pass the address of LIB$PUT_OUTPUT and LIB$GET_INPUT, they must be declared as EXTERNAL. You may supply your own input and/or output routines.

```
1       *                                                HELPOUT.COB
2        IDENTIFICATION DIVISION.
3        *
4        PROGRAM-ID. HELPOUT.
5        *
6        *   This program satisfies an initial help request
7        *   and enters interactive HELP mode.  The library
8        *   used is SYS$HELP:HELPLIB.HLB.
9        *
10       DATA DIVISION.
11       WORKING-STORAGE SECTION.
12
13       01  HELP_KEY          PIC X(32).
14    ❷  01  LIB$PUT_OUTPUT    PIC S9(9) COMP  VALUE EXTERNAL LIB$PUT_OUTPUT.
15    ❷  01  LIB$GET_INPUT     PIC S9(9) COMP  VALUE EXTERNAL LIB$GET_INPUT.
16       01  STAT              PIC S9(9) COMP.
17
18       PROCEDURE DIVISION.
19       BEGIN.
20       *
21       *   Request a HELP key
22           DISPLAY 'What topic would you like HELP with? ' WITH NO ADVANCING.
23           ACCEPT HELP_KEY.
24       *
25       *   Locate and print the HELP text
26    ❶      CALL 'LBR$OUTPUT_HELP' USING BY VALUE LIB$PUT_OUTPUT
27                                        BY VALUE 0
28                                        BY DESCRIPTOR HELP_KEY 'HELPLIB'
29                                        BY VALUE 0 LIB$GET_INPUT
30                                GIVING STAT.
31           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
32           STOP RUN.
```

Example 1 Program Using Help Libraries in Prompting Mode (Sheet 1 of 2)

```
$ COBOL HELPOUT
$ LINK HELPOUT
$ RUN HELPOUT
What topic would you like HELP with? TYPE

TYPE


    Displays the contents of a file or group of  files  on  the  current
    output device.

    Format:

      TYPE file-spec[,...]

  Additional information available:

  Parameters Command_Qualifiers
  /BACKUP    /BEFORE    /BY_OWNER  /CONFIRM   /CREATED   /EXCLUDE   /EXPIRED
  /MODIFIED  /OUTPUT    /SINCE
  Examples

TYPE Subtopic? /OUTPUT

TYPE

  /OUTPUT

   /OUTPUT=file-spec

    Requests that the output from the TYPE command  be  written  to  the
    specified  file,  rather  than to the current default output device,
    SYS$OUTPUT.


TYPE Subtopic? ^Z
$
```

Example 1 Program Using Help Libraries in Prompting Mode (Sheet 2 of 2)

Example 2

Example 2 shows a command description file and the associated image file.

❶ The successive calls to CLI$GET_VALUE will return the value entered on the command line to the image. Note that command line syntax errors would be detected before the image is invoked.

❷ After the information has been received, the full name is displayed.

```
1       *                                                NAME.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. NAME.
5       *
6       *   This program will obtain information from a
7       *   command line and process it.
8       *
9       DATA DIVISION.
10      WORKING-STORAGE SECTION.
11
12      01  FIRST_NAME      PIC X(20).
13      01  MIDDLE_NAME     PIC X(20).
14      01  LAST_NAME       PIC X(20).
15      01  STAT            PIC S9(9) COMP.
16      01  CLI$_ABSENT     PIC S9(9) COMP
17                          VALUE EXTERNAL CLI$_ABSENT.
18
19      PROCEDURE DIVISION.
20      BEGIN.
21      *
22      *   Get values for parameters and qualifiers
23          CALL 'CLI$GET_VALUE' USING BY DESCRIPTOR 'FIRST' FIRST_NAME
24                  GIVING STAT.
25          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
26          CALL 'CLI$GET_VALUE' USING BY DESCRIPTOR 'LAST' LAST_NAME
27                  GIVING STAT.
28   ❶      IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
29          CALL 'CLI$GET_VALUE' USING BY DESCRIPTOR 'MIDDLE' MIDDLE_NAME
30                  GIVING STAT.
31          IF STAT = CLI$_ABSENT THEN
32                  MOVE SPACES TO MIDDLE_NAME
33          ELSE
34                  IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
35      *
36      *   Process parameters and qualifiers
37          IF MIDDLE_NAME NOT = SPACES THEN
38                  DISPLAY '      '  FIRST_NAME
39                  DISPLAY '      '  MIDDLE_NAME
40   ❷              DISPLAY '      '  LAST_NAME
41          ELSE
42                  DISPLAY '      '  FIRST_NAME
43                  DISPLAY '      '  LAST_NAME.
44          STOP RUN.
```

Example 2 Creating User-Defined Commands (Sheet 1 of 2)

```
    1       !                                                   NAME.CLD
    2       DEFINE VERB name
    3           IMAGE my_directory:name
    4           PARAMETER p1, PROMPT='First Name',
    5                         LABEL=first,
    6                         VALUE(REQUIRED)
    7           PARAMETER p2, PROMPT='Last Name',
    8                         LABEL=last,
    9                         VALUE(DEFAULT='student')
   10           QUALIFIER middle, VALUE

$ COBOL NAME
$ LINK NAME
$ ASSIGN SYS$DISK:[COURSE.PROG.FOR.INTR] MY_DIRECTORY
$ SET COMMAND MY_DIRECTORY:NAME.CLD
$
$ NAME/MIDDLE=THE KERMIT FROG
     KERMIT
     THE
     FROG

$ NAME/MIDDLE=S.
_First Name: SCOOTER
_Last Name: SMITH
     SCOOTER
     S.
     SMITH

$ NAME
_First Name: SCOOTER
_Last Name: SMITH
     SCOOTER
     SMITH

$ NAME
_First Name: SCOOTER
_Last Name:
     SCOOTER
     student
$
```

Example 2 Creating User-Defined Commands (Sheet 2 of 2)

Example 3

Example 3 illustrates a user-defined command language interface. The CLI interface routines are used for parsing user input.

❶ The command tables are defined using the command definition language. Command description files designed for processing application-supplied strings specify routine names, rather than image names. COBOL subroutine names are specified to process the application-defined commands.

❷ LIB$GET_INPUT retrieves the contents of the command line.

❸ CLI$DCL_PARSE validates the syntax of the input line. If a syntax error is detected, DCL will supply an error message and the user will be prompted for another command.

❹ CLI$DISPATCH calls a subroutine to execute the command. The routine names are defined in the command tables.

❺ If the syntax is valid, the appropriate routine is performed. The two possible routines are REPORT_COMMAND and EXIT_COMMAND.

❻ REPORT_COMMAND obtains the file name, determines if the /EDIT qualifier is present, and prints the file.

❼ If the /EDIT quailfier is present, EDIT_QUALIFIER is called. The program invokes the EDT editor as a subprocess using LIB$SPAWN. (Using LIB$SPAWN to create subprocesses to execute DCL commands is a topic of the Creating and Managing Other Processes module.)

❽ LIB$SPAWN is also used to print the file.

❾ To run the program, the command file TEST.CLD must be compiled by the command definition utility. Link the resulting .OBJ file with COMMAND.OBJ before running COMMAND.

The /NOTRACE option is specified on the LINK command to suppress the printing of traceback information when a user enters an invalid command.

```
1       *                                                COMMAND.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. COMMAND.
5       *
6       *   This program contains a command language, and uses
7       *   the command language interface routines to parse
8       *   and process the commands.
9       *
10      DATA DIVISION.
11      WORKING-STORAGE SECTION.
12
13      01  COMMAND_LINE    PIC X(50).
14      01  PROMPT_STRING   PIC X(7)      ' VALUE 'TEST> '.
15      01  STAT            PIC S9(9) COMP.
16      01  TEST_TABLES     PIC S9(9) COMP  VALUE EXTERNAL TEST_TABLES.
17
18      PROCEDURE DIVISION.
19      BEGIN.
20      *
21      *   Get the input line
22  (2)     CALL 'LIB$GET_INPUT' USING BY DESCRIPTOR COMMAND_LINE
23                                                   PROMPT_STRING
24                               GIVING STAT.
25          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
26      *
27      *   Check for valid command syntax
28  (3)     CALL 'CLI$DCL_PARSE' USING BY DESCRIPTOR COMMAND_LINE
29                                     BY VALUE TEST_TABLES
30                               GIVING STAT.
31          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
32      *
33      *   Dispatch to the appropriate routine
34  (4)     CALL 'CLI$DISPATCH'.
35          GO TO BEGIN.
36      END PROGRAM COMMAND.
37
38
39      IDENTIFICATION DIVISION.
40      *
41  (5) PROGRAM-ID. REPORT_COMMAND.
42      *
43      DATA DIVISION.
44      WORKING-STORAGE SECTION.
45
46      01  PRINT_COMMAND   PIC X(21).
47      01  PRINT_VERB      PIC X(6)        VALUE 'PRINT '.
48      01  FILE_NAME       PIC X(15).
49      01  STAT            PIC S9(9) COMP.
```

Example 3 Application Program with a Command Language (Sheet 1 of 3)

```
50
51      PROCEDURE DIVISION.
52      BEGIN.
53      *
54      *   Retrieve the file specification
55  6       CALL 'CLI$PRESENT' USING BY DESCRIPTOR 'FILESPEC'
56                             GIVING STAT.
57          IF STAT IS SUCCESS
58             CALL 'CLI$GET_VALUE' USING BY DESCRIPTOR 'FILESPEC'
59                                                     FILE_NAME.
60          CALL 'CLI$PRESENT' USING BY DESCRIPTOR 'EDIT'
61                             GIVING STAT.
62          IF STAT IS SUCCESS
63             CALL 'EDIT_QUALIFIER' USING FILE_NAME.
64      *
65      *   Print the file
66          STRING PRINT_VERB DELIMITED BY ' '
67                 ' ' DELIMITED BY SIZE
68                 FILE_NAME DELIMITED BY SIZE
69                 INTO PRINT_COMMAND.
70  8       CALL 'LIB$SPAWN' USING BY DESCRIPTOR PRINT_COMMAND
71                               BY VALUE 0 0 0
72                               BY DESCRIPTOR 'MYSUB'
73                               BY VALUE  0 0 0 0 0
74                           GIVING STAT.
75          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
76          EXIT PROGRAM.
77      END PROGRAM REPORT_COMMAND.
78
79
80
81      IDENTIFICATION DIVISION.
82      *
83  5   PROGRAM-ID. EXIT_COMMAND.
84      *
85      DATA DIVISION.
86      WORKING-STORAGE SECTION.
87      PROCEDURE DIVISION.
88      BEGIN.
89          STOP RUN.
90      END PROGRAM EXIT_COMMAND.
91
92
93
94
95      IDENTIFICATION DIVISION.
96      *
97      PROGRAM-ID. EDIT_QUALIFIER.
98      *
```

Example 3 Application Program with a Command Language (Sheet 2 of 3)

```
 99        DATA DIVISION.
100
101        WORKING-STORAGE SECTION.
102        01  EDIT_COMMAND     PIC X(21).
103        01  EDIT_VERB        PIC X(6)        VALUE 'EDIT '.
104        01  STAT             PIC S9(9) COMP.
105
106        LINKAGE SECTION.
107        01  FILE_NAME        PIC X(15).
108
109        PROCEDURE DIVISION USING FILE_NAME.
110        BEGIN.
111        *
112        *   Invoke EDT
113            STRING EDIT_VERB DELIMITED BY ' '
114                   ' ' DELIMITED BY SIZE
115                   FILE_NAME DELIMITED BY SIZE
116                   INTO EDIT_COMMAND.
117     7      CALL 'LIB$SPAWN' USING BY DESCRIPTOR EDIT_COMMAND
118                                   BY VALUE 0 0 0
119                                   BY DESCRIPTOR 'MYSUB'
120                                   BY VALUE  0 0 0 0 0
121                               GIVING STAT.
122            IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
123            EXIT PROGRAM.
124        END PROGRAM EDIT_QUALIFIER.


  1     1  !                                                TEST.CLD
  2            MODULE test_tables
  3            DEFINE VERB report
  4                    ROUTINE report_command
  5                    PARAMETER p1, LABEL = filespec
  6                    QUALIFIER edit
  7            DEFINE VERB exit
  8                    ROUTINE exit_command

$ COBOL COMMAND
$ SET COMMAND/OBJECT TEST
9 $ LINK/NOTRACE COMMAND,TEST
$ RUN COMMAND
TEST>  REPORT/EDIT TODAY.DAT
Input file does not exist
[EOB]
*I
            This is a test of calling EDT from a program.
            This file will print upon exit from EDT.

[EOB]
*exit
WORK1:[COURSE.V4PROG.COB.INTR]TODAY.DAT;1 2 lines
  Job TODAY (queue SYS$PRINT, entry 162) started on LPA0
TEST>  exit
$
```

Example 3 Application Program with a Command Language (Sheet 3 of 3)

Example 4

Example 4 signals errors that are user-defined rather than system defined. The message file contains a message source file corresponding to these errors.

❶ This directive defines a customer facility PRG with a customer facility code 1.

❷ These directives define the message text for the following messages:

```
%PRG-I-BEGIN
%PRG-I-END
```

The message text includes formatted ASCII output.

❸ These directives define the message text for the following messages:

```
%PRG-F-FATAL
%PRG-W-WARN
```

❹ The global symbol PRG__BEGIN is defined by MESTXT and represents the status code that is translated to %PRG-I-BEGIN. The other arguments of LIB$SIGNAL are used for the formatted ASCII output of the message text.

❺ The global symbol PRG__FATAL is defined by MESTXT and represents the status code that is translated to the status message %PRG-F-FATAL.

❻ To run this program, the message MESTXT must be compiled by the message utility. Link the resulting .OBJ file with MESPRG.OBJ before running MESPRG.

❼ The messages are output when the program is run, and can be dynamic. The following substitutions were made from the message text file:

!AS - program name MESPRG
!%D - date and time

```
 1       *                                                MESPRG.COB
 2       IDENTIFICATION DIVISION.
 3       *
 4       PROGRAM-ID. MESPRG.
 5       *
 6       *   This program illustrates signalling and the use
 7       *   of user defined error messages.  It requires the
 8       *   message file MESTXT.MSG.
 9       *
10       DATA DIVISION.
11       WORKING-STORAGE SECTION.
12
13       01  PROC_NAME       PIC X(6)        VALUE 'MESPRG'.
14       01  PRG_BEGIN       PIC S9(9) COMP  VALUE EXTERNAL  PRG_BEGIN.
15       01  PRG_FATAL       PIC S9(9) COMP  VALUE EXTERNAL  PRG_FATAL.
16
17       PROCEDURE DIVISION.
18       BEGIN.
19       *
20       *   Signal an informational message
21  (4)      CALL 'LIB$SIGNAL' USING BY VALUE PRG_BEGIN 2
22                                   BY DESCRIPTOR PROC_NAME
23                                   BY VALUE 0.
24       *
25       *   Signal a fatal message
26  (5)      CALL 'LIB$SIGNAL' USING BY VALUE PRG_FATAL.
27       *
28           DISPLAY 'The program never gets this far.'
29       *
30           STOP RUN.

 1       !                                                MESTXT.MSG
 2           .TITLE  MESTXT
 3       !
 4       !   This is the message file for the program MESPRG
 5
 6  (1)      .FACILITY       PRG, 1
 7
 8  (2)     {.SEVERITY       INFO
 9          {BEGIN   <Beginning procedure !AS, at !%D.> /FAO=2
10          {END     <Ending procedure !AS at !%D.> /FAO=2
11
12  (3)     {.SEVERITY       SEVERE
13          {FATAL   <This is fatal.>
14
15           .END

$ COBOL MESPRG
(6) {$ MESSAGE/OBJECT MESTXT
    {$ LINK MESPRG, MESTXT
$ RUN MESPRG
(7) {%PRG-I-BEGIN, Beginning procedure MESPRG, at 27-FEB-1984 16:37:42.03.
    {%PRG-F-FATAL, This is fatal.
%TRACE-F-TRACEBACK, symbolic stack dump follows
module name     routine name                     line       rel PC    abs PC

MESPRG          MESPRG                             26     0000002C  0000062C
$
```

Example 4 Generating User-Defined Error Messages

Lab Exercises

1. Use an editor to create the help files ENOUGH.HLP and REMOVE.HLP shown below. ENOUGH AND REMOVE are fictitious DCL commands. Create a help library containing the two files and modify Example 1 (HELPOUT.COB in this module) to access the library you create.

```
ENOUGH.HLP

 1 ENOUGH
```

The ENOUGH command is reserved for those times that you intend to quit your job. It does the following:

- deletes all your files
- deletes your UAF record
- sends a copy of your letter of resignation to specified users
- optionally prints n copies of your resume
- logs you off

Format:

```
ENOUGH[/RESUME=n] [user_name,...]

2 user_name
```

specifies the name of the user or users to whom you wish to send a letter of resignation.

```
2 /RESUME=n
```

Specifies the number of copies of your resume to be printed on the system default printer.

```
REMOVE.HLP

1 REMOVE
```

The REMOVE command is a system management command used to eliminate all traces of a particular user.

Format:

```
REMOVE user_name

2 user_name
```

The user_name specifies the user to be removed from the system.

2. Modify Example 4 in this module (MESPRG.COB and MESTXT.MSG) to include a warning message. MESTXT.MSG should be modified to include the warning message, "Warning duplicate file name". MESPRG.COB should be modified to signal the warning message.

3. Use the Command Definition Utility to define a new DCL command (LIST) that accepts one required parameter. Write a program LABSOL3.COB that processes the command in the following way:

   - Lists the odd positive integers less than 100, if the parameter entered is the word ODD
   - Lists the even positive integers less than 100, if the parameter entered is the word EVEN

Lab Solutions

1.

```
1       *                                                LABSOL1.COB
2        IDENTIFICATION DIVISION.
3        *
4        PROGRAM-ID. LABSOL1.
5        *
6        *   This program satisfies an initial help request
7        *   and enters interactive HELP mode.  The library
8        *   used is specified by the user.
9        *
10       DATA DIVISION.
11       WORKING-STORAGE SECTION.
12
13       01  HELP_KEY         PIC X(32).
14       01  LIB$PUT_OUTPUT   PIC S9(9) COMP  VALUE EXTERNAL LIB$PUT_OUTPUT.
15       01  LIB$GET_INPUT    PIC S9(9) COMP  VALUE EXTERNAL LIB$GET_INPUT.
16       01  STAT             PIC S9(9) COMP.
17
18       PROCEDURE DIVISION.
19       BEGIN.
20       *
21       *   Request a HELP key
22           DISPLAY 'What topic would you like HELP with?
23                   WITH NO ADVANCING.
24           ACCEPT HELP_KEY.
25       *
26       *   Locate and print the HELP text
27           CALL 'LBR$OUTPUT_HELP' USING BY VALUE LIB$PUT_OUTPUT
28                                        BY VALUE 0
29                                        BY DESCRIPTOR HELP_KEY
30                                             'DISK$COURSE:[USER]MYLIB'
31                                        BY VALUE 0 LIB$GET_INPUT
32                                   GIVING STAT.
33           IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
34           STOP RUN.
```

2.

```
1       *                                                LABSOL2A.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL2A.
5       *
6       *   This program illustrates signalling and the use
7       *   of user defined error messages.  It requires the
8       *   message file MESTXT.MSG.
9       *
10      DATA DIVISION.
11      WORKING-STORAGE SECTION.
12
13      01  PROC_NAME       PIC X(8)        VALUE 'LABSOL2A'.
14      01  PRG_BEGIN       PIC S9(9) COMP  VALUE EXTERNAL PRG_BEGIN.
15      01  PRG_FATAL       PIC S9(9) COMP  VALUE EXTERNAL PRG_FATAL.
16      01  PRG_WARN        PIC S9(9) COMP  VALUE EXTERNAL PRG_WARN.
17
18      PROCEDURE DIVISION.
19      BEGIN.
20      *
21      *   Signal an informational message
22          CALL 'LIB$SIGNAL' USING BY VALUE PRG_BEGIN 2
23                                  BY DESCRIPTOR PROC_NAME
24                                  BY VALUE 0.
25      *
26      *   Signal a warning message
27          CALL 'LIB$SIGNAL' USING BY VALUE PRG_WARN.
28      *
29      *   Signal a fatal message
30          CALL 'LIB$SIGNAL' USING BY VALUE PRG_FATAL.
31      *
32      *
33          STOP RUN.
```

```
1       !                                                LABSOL2B.MSG
2           .TITLE  MESTXT
3       !
4       !   This is the message file for the program LABSOL2A
5
6           .FACILITY       PRG, 1
7
8           .SEVERITY       INFO
9           BEGIN   <Beginning procedure !AS, at !%D.> /FAO=2
10          END     <Ending procedure !AS at !%D.> /FAO=2
11
12          .SEVERITY       WARNING
13          WARN            <Warning duplicate filename>
14          .SEVERITY       SEVERE
15          FATAL   <This is fatal.>
16
17          .END
```

3.

```
1       *                                                    LABSOL3A.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL3A.
5       *
6       *   This program processes the command LIST. It lists the
7       *   positive integers less than 100. If the command LIST ODD
8       *   is entered, it lists odd integers less than 100. If the
9       *   command LIST EVEN is entered, it lists even integers less
10      *   than 100.
11      *
12      DATA DIVISION.
13
14      WORKING-STORAGE SECTION.
15
16      01  PARA            PIC X(20).
17      01  I               PIC 999.
18      01  DISP_I          PIC ZZ9.
19
20      PROCEDURE DIVISION.
21      BEGIN.
22      *
23          CALL 'CLI$GET_VALUE' USING BY DESCRIPTOR 'PARAM' PARA.
24          DISPLAY PARA
25          IF PARA IS EQUAL TO 'ODD' THEN
26             PERFORM VARYING I FROM 1 BY 2 UNTIL I > 99
27               MOVE I TO DISP_I
28               DISPLAY I
29             END-PERFORM
30          ELSE
31                  IF PARA IS EQUAL TO 'EVEN' THEN
32                     PERFORM VARYING I FROM 2 BY 2 UNTIL I > 98
33                       MOVE I TO DISP_I
34                       DISPLAY I
35                     END-PERFORM
36                  END-IF
37          END-IF
38          STOP RUN.
```

```
1       DEFINE VERB list
2           IMAGE disk$course:[user]list
3           PARAMETER p1, LABEL=param,
4                               PROMPT="parameter?",
5                         VALUE(REQUIRED)
```

```
1       DEFINE VERB list
2           IMAGE disk$course:[user]list
3           PARAMETER p1, LABEL=param,
4                               PROMPT="parameter?",
5                         VALUE(REQUIRED,TYPE=KEYS)
6
7       DEFINE TYPE KEYS
8               KEYWORD ODD
9               KEYWORD EVEN
```

Creating, Accessing, and Ordering Files

File Locations

On-line files are provided to help you master this module. The files are located in the directory:

```
DISK$COURSE:[COURSE.V4PROG.COB.FILE]
```

For your convenience, your system manager may have created the following logical name equivalence:

```
DISK$COURSE:[COURSE.V4PROG.COB.FILE] = V4PROG$COB$FILE
```

Two types of files are provided:

1. The Module Programming Examples, which are also listed in this language workbook.
2. Solutions to the Laboratory Exercises.

If you wish to modify and/or compile, link, and run any of the programs, first copy the file into your directory.

The VAX COBOL programmer accesses VAX RMS files through COBOL I/O statements, such as OPEN, CLOSE, READ, and WRITE. Table 1 compares the VAX RMS and VAX COBOL I/O directives. For a discussion of these, and other COBOL I/O statements, see the *VAX COBOL Language Reference Manual.*

Table 1 VAX RMS and VAX COBOL I/O Directives

Operation	VAX RMS Directive	VAX COBOL Statement
File Processing		
Create a file	$CREATE	OPEN OUTPUT
Access an existing file	$OPEN	OPEN INPUT or I-O
Terminate access to a file	$CLOSE	CLOSE
Record Processing		
Access a record stream	$CONNECT	OPEN
Terminate access to a record stream	$DISCONNECT	CLOSE
Retrieve a record	$GET	READ
Write a record	$PUT	WRITE
Locate a record	$FIND	FIND, FETCH
Remove a record from a file	$DELETE	DELETE
Rewrite an existing record	$UPDATE	REWRITE
Release access to a locked record	$FREE	UNLOCK

COBOL programmers cannot access the $FAB and $RAB control blocks directly. The FILE-CONTROL paragraph of a COBOL program sets up the $FAB and $RAB control blocks.

The APPLY clause in the I-O-CONTROL paragraph allows communication with RMS that is not otherwise possible in COBOL. The phrases of the APPLY clause can be used to perform I/O processing.

LEARNING ACTIVITIES

1. Refer to the *VAX/VMS Record Management Services Reference Manual* for more information about RMS file and record access.
2. Refer to the *VAX COBOL User's Guide* for more information about COBOL I/O processing.

Example 1

Example 1 uses logical names to perform I/O operations to a file or a terminal.

❶ Before a file can be opened, appropriate entries must be made in a FILE-CONTROL paragraph for each file to be accessed. The SELECT clause determines the name of the file used in the COBOL program. The ASSIGN clause determines the name of the file as it exists on the external medium, SYS$DISK:[COURSE.V4PROG.COB.FILE]OUT.DAT, for example. The ASSIGN may also be used to designate a logical name that will be translated at run time.

The INPUT_FILE is assigned to the logical name COB$INPUT. By assigning COB$INPUT, you can accept input from a file or a terminal:

```
$ ASSIGN file_specification COB$INPUT
```

```
$ ASSIGN TTxx: COB$INPUT
```

The OUTPUT_FILE is written to the file OUT.DAT, unless a logical name assignment is made. To send output to your terminal, issue the following command:

```
$ ASSIGN TTxx: OUT
```

❷ The I-O-CONTROL paragraph provides phrases that you can use to improve I/O performance. Using the SAME RECORD AREA reduces address space and processing overhead.

❸ The files must be opened prior to performing any operations on them.

❹ Read and write operations are performed. Since the files share the same record area, moving the record from INPUT LINE to OUTPUT LINE is unnecessary.

❺ After processing operations are completed, close the files. All opened files are closed upon image exit.

```
1       *                                                TERMNALIO.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. TERMNALIO.
5       *
6       *   This program illustrates the use of logical names
7       *   to direct input and output operations to a terminal
8       *   or a file.
9       *
10      ENVIRONMENT DIVISION.
11      INPUT-OUTPUT SECTION.
12  (1) FILE-CONTROL.
13
14          SELECT INPUT_FILE ASSIGN TO 'COB$INPUT'
15                  FILE STATUS IS IOSTAT.
16          SELECT OUTPUT_FILE ASSIGN TO 'OUT'.
17
18  (2) I-O-CONTROL.
19
20          SAME RECORD AREA FOR INPUT_FILE OUTPUT_FILE.
21
22      DATA DIVISION.
23      FILE SECTION.
24
25      FD  INPUT_FILE
26                  RECORD VARYING DEPENDING ON TEXT_LENGTH.
27      01  INPUT_LINE              PIC X(80).
28
29      FD  OUTPUT_FILE
30                  RECORD VARYING DEPENDING ON TEXT_LENGTH.
31      01  OUTPUT_LINE             PIC X(80).
32
33      WORKING-STORAGE SECTION.
34
35      01  TEXT_LENGTH             PIC 9(9) COMP.
36      01  IOSTAT                  PIC XX.
37          88      END_OF_FILE              VALUE '13'.
38
39      PROCEDURE DIVISION.
40      BEGIN.
41  (3)     OPEN INPUT INPUT_FILE.
42          OPEN OUTPUT OUTPUT_FILE.
43          DISPLAY 'Enter data record: ' WITH NO ADVANCING.
44          READ INPUT_FILE AT END CONTINUE END-READ
45          PERFORM UNTIL END_OF_FILE
46                  DISPLAY ' '
47  (4)             WRITE OUTPUT_LINE
48                  DISPLAY 'Enter data record: ' WITH NO ADVANCING
49                  READ INPUT_FILE AT END CONTINUE END-READ
50          END-PERFORM.
51  (5)     CLOSE INPUT_FILE OUTPUT_FILE.
52          STOP RUN.
```

Example 1 Using Logical Names (Sheet 1 of 2)

```
$ COBOL TERMNALIO
$ LINK TERMNALIO
$ ASSIGN TTE7: COB$INPUT
$ RUN TERMNALIO
Enter data record: ENTERING DATA

Enter data record: MORE DATA

Enter data record: ^Z

$ TYPE OUT.DAT
ENTERING DATA
MORE DATA

$ ASSIGN TTE7: OUT
$ RUN TERMNALIO
Enter data record: FIRST RECORD

FIRST RECORD
Enter data record: SECOND RECORD

SECOND RECORD
Enter data record: LAST RECORD

LAST RECORD
Enter data record: ^Z
$
$
```

Example 1 Using Logical Names (Sheet 2 of 2)

Example 2

Each record in a relative file is assigned to a specific cell in that file. On sequential WRITEs, the records are written to consecutive empty cells. Random WRITEs place the records into cell numbers as provided by the RECORD KEY clause in the File Description entry. Example 2 randomly accesses a relative file.

❶ These statements define the file and record processing characteristics. Although a relative file organization is specified, RMS would, in fact, obtain the file organization from an existing file. If the file's organization were not relative, the file open statement would fail.

❷ The random READ statement reads the record specified in REC_NUM (as opposed to a sequential READ statement, which would read the next consecutive record). Because of the FILE STATUS clause in the FILE-CONTROL section, the status code for the record operation is returned in the variable IOSTAT.

❸ These statements test the record operation status code returned by the READ.

```
 1       *                                                RELATIVE.COB
 2       IDENTIFICATION DIVISION.
 3       *
 4       PROGRAM-ID. RELATIVE_PROGRAM.
 5       *
 6       *   This program illustrates accessing a relative file
 7       *   randomly.   It also performs some I/O status checks.
 8       *
 9       ENVIRONMENT DIVISION.
10       INPUT-OUTPUT SECTION.
11       FILE-CONTROL.
12
13           SELECT RELATIVE_FILE ASSIGN TO 'REL.DAT'
14                   ORGANIZATION IS RELATIVE
15                   FILE STATUS IS IOSTAT.
16
17    (1)DATA DIVISION.
18
19       FILE SECTION.
20       FD  RELATIVE_FILE
21           ACCESS MODE IS RANDOM
22           RELATIVE KEY IS REC_NUM.
23       01  RECERD          PIC X(20).
24
25       WORKING-STORAGE SECTION.
26       01  REC_NUM         PIC 999.
27       01  IOSTAT          PIC XX.
28           88 SUCCESSFUL          VALUE '00'.
29           88 REC_NOT_FOUND       VALUE '23'.
30
31       PROCEDURE DIVISION.
32       DECLARATIVES.
33       I-O-ERROR SECTION.
34           USE AFTER STANDARD ERROR PROCEDURE ON RELATIVE_FILE.
35       CHECK-ERROR-CODES.
36           IF REC_NOT_FOUND
37    (3)            DISPLAY 'Nonexistant record.'
38           ELSE
39                   CALL 'LIB$SIGNAL' USING BY VALUE RMS_STS RMS_STV
40           END-IF.
41       END DECLARATIVES.
42
43       RECORD-ACCESS SECTION.
44       BEGIN.
45       *
46       *   Get records by record number until e-o-f
47       *           Prompt for record number
48           OPEN INPUT RELATIVE_FILE.
49
```

Example 2 Using a Relative File (Sheet 1 of 2)

```
   50        GET-NEXT-RECORD.
   51            DISPLAY 'Record number: ' WITH NO ADVANCING.
   52            ACCEPT REC_NUM AT END STOP RUN.
   53        *
   54        *   Read record by record number
   55            INITIALIZE RECERD.
   56    ❷       READ RELATIVE_FILE.
   57            IF SUCCESSFUL DISPLAY RECERD.
   58            GO TO GET-NEXT-RECORD.

$ COBOL RELATIVE
$ LINK RELATIVE
$ RUN RELATIVE
Record number: 007
08001FLANJE119PL1920
Record number: 004
07850DALLJE119FI1050
Record number: 032
Nonexistant record.
Record number: ^Z
$
$
```

Example 2 Using a Relative File (Sheet 2 of 2)

Example 3

Example 3 accesses indexed files.

❶ To allow random by key access to an indexed file, the ACCESS MODE parameter of the SELECT statement must be set to DYNAMIC. This allows the access mode to change depending on the format of the READ statement.

❷ A random READ is issued to locate a record by key value.

❸ A sequential read is performed to get the next record in the correct key of reference by issuing a READ NEXT RECORD. VAX RMS will not return a status code when the key value changes. Consequently, the program must check the key value in the record to determine whether or not it has changed.

```
1       *                                                INDEXED.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. INDEXED_PROGRAM.
5       *
6       *   This program illustrates random and sequential
7       *   access of an indexed sequential file.  It prompts
8       *   for a department number then prints the name of
9       *   each person in the department.
10      *
11      ENVIRONMENT DIVISION.
12      INPUT-OUTPUT SECTION.
13      FILE-CONTROL.
14
15  (1)     SELECT INDEXED_FILE ASSIGN TO 'IDX'
16                 ORGANIZATION IS INDEXED
17                 ACCESS MODE IS DYNAMIC
18                 RECORD KEY IS DEPT
19                 FILE STATUS IS IOSTAT.
20
21      DATA DIVISION.
22
23      FILE SECTION.
24      FD  INDEXED_FILE.
25      01  INPUT_BUFFER.
26          02       FILLER                  PIC X(11).
27          02       DEPT                    PIC XXX.
28          02       FILLER                  PIC X(6).
29
30      WORKING-STORAGE SECTION.
31      01  DEPT_KEY                 PIC XXX.
32      01  EMPLOYEE_RECORD.
33          02       ID_NUM                  PIC X(5).
34          02       EMP_NAME                PIC X(6).
35          02       EMP_DEPT                PIC XXX.
36          02       SKILL                   PIC XX.
37          02       SALARY                  PIC XXXX.
38      01  IOSTAT                   PIC XX.
39
40      PROCEDURE DIVISION.
41      BEGIN.
42      *
43      *   Open indexed file
44          OPEN INPUT INDEXED_FILE.
45      *
46      *   Obtain department number
47      GET-NEXT-DEPARTMENT.
48          DISPLAY 'Enter department no. (119,210,220): '
49                   WITH NO ADVANCING.
50          ACCEPT DEPT_KEY AT END STOP RUN.
51          MOVE DEPT_KEY TO DEPT.
52      *
53      *   Read first DEPT_KEY record
54  (2)     READ INDEXED_FILE INTO EMPLOYEE_RECORD
55             INVALID KEY DISPLAY 'Invalid department number.'
56             GO TO GET-NEXT-DEPARTMENT.
57      *
58      *   Read next DEPT_KEY record
59          PERFORM UNTIL DEPT_KEY NOT EQUAL DEPT
60             DISPLAY EMP_NAME
61             READ INDEXED_FILE NEXT RECORD INTO EMPLOYEE-RECORD
62  (3)           AT END MOVE 999 TO DEPT
63             END-READ
64          END-PERFORM.
65          GO TO GET-NEXT-DEPARTMENT.
```

Example 3 Using an Indexed File (Sheet 1 of 2)

```
$ COBOL INDEXED
$ LINK INDEXED
$ RUN INDEXED
Enter department no. (119,210,220): 119
ANDEWF
DALLJE
FLANJE
FLINGA
GREEJW
JONEKB
MANKCA
MARSJJ
REDFBB
SCHAWE
WIENSH
Enter department no. (119,210,220): 011
Invalid department number.
Enter department no. (119,210,220): ^Z
$
```

Example 3 Using an Indexed File (Sheet 2 of 2)

Example 4

Although it is possible to create files of any organization from within a COBOL program, it may not be desirable. A fairly good understanding of RMS is required to be able to design and create efficient RMS files. Furthermore, COBOL does not allow you to take full advantage of the flexibility RMS provides for defining file attributes. The RMS utilites were designed to simplify the task of creating efficient files.

Example 4 creates a sequential file from an existing FDL file.

❶ The data file SEQ.DAT is created using the attributes of SEQ.FDL. Note that trailing optional arguments **must** be omitted on the FDL$CREATE create call.

❷ FDL$CREATE closes the file after creation, so SEQ.DAT must be opened EXTENDed. If an OPEN INPUT were issued instead, a new version of the file would be created with default attributes rather than those specified in SEQ.FDL.

❸ The file is populated from an existing sequential file.

❹ These files share the same memory area for the record descriptions of each file. Therefore, it is not necessary to move the record from INPUT_RECORD to OUTPUT_RECORD before writing to the new file. This improves file processing time and also saves memory.

❺ When all the records have been entered, the files are closed.

```
1       *                                                SEQUENTIAL.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. SEQUENTIL.
5       *
6       *   This program will create and populate a sequential file.
7       *
8       ENVIRONMENT DIVISION.
9       INPUT-OUTPUT SECTION.
10      FILE-CONTROL.
11
12          SELECT SEQUENTIAL_FILE ASSIGN TO 'SEQ'
13                  ORGANIZATION IS SEQUENTIAL.
14
15          SELECT EMPLOYEE_FILE ASSIGN TO 'EMPLOYEE'
16                  ORGANIZATION IS SEQUENTIAL.
17
18      I-O-CONTROL.
19          SAME RECORD AREA FOR SEQUENTIAL_FILE EMPLOYEE_FILE.
20
21      DATA DIVISION.
22      FILE SECTION.
23
24      FD  EMPLOYEE_FILE
25          ACCESS MODE IS SEQUENTIAL.
26      01  INPUT_RECORD            PIC X(20).
27
28      FD  SEQUENTIAL_FILE
29          ACCESS MODE IS SEQUENTIAL.
30      01  OUTPUT_RECORD           PIC X(20).
31
32      WORKING-STORAGE SECTION.
33      01  STAT            PIC S9(9) COMP.
34
35      PROCEDURE DIVISION.
36      BEGIN.
37      *
38      *   Create a sequential file
39  (1)     CALL 'FDL$CREATE' USING BY DESCRIPTOR 'SEQ.FDL'
40                                                'SEQ.DAT'
41                          GIVING STAT.
42          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
43      *
44      *   Open the files
45  (2)     OPEN EXTEND SEQUENTIAL_FILE.
46  (3)     OPEN INPUT  EMPLOYEE_FILE.
47      *
48      *   Read a record
49          READ EMPLOYEE_FILE AT END
50  (5)                         CLOSE EMPLOYEE_FILE
51                                    SEQUENTIAL_FILE
52                              STOP RUN.
53      GET-NEXT-RECORD.
54      *
55      *   Since the same record area is used it is not necesary to
56      *   move INPUT_RECORD to OUTPUT_RECORD.
57  (4)        WRITE OUTPUT_RECORD.
58             DISPLAY '    ' OUTPUT_RECORD.
59             READ EMPLOYEE_FILE AT END STOP RUN.
60             GO TO GET-NEXT-RECORD.
```

Example 4 Creating a Sequential File (Sheet 1 of 2)

```
     1        IDENT        " 2-FEB-1984 14:24:18   VAX-11 FDL Editor"
     2
     3        SYSTEM
     4            SOURCE                    VAX/VMS
     5
     6        FILE
     7            ALLOCATION                3
     8            BEST_TRY_CONTIGUOUS       no
     9            EXTENSION                 0
    10            NAME                      "SEQ.DAT"
    11            ORGANIZATION              sequential
    12
    13        RECORD
    14            BLOCK_SPAN                yes
    15            CARRIAGE_CONTROL          carriage_return
    16            FORMAT                    variable
    17            SIZE                      0
$ COBOL SEQUENTIAL
$ LINK SEQUENTIAL
$ RUN SEQUENTIAL
   07672ALBEHA210SE2100
   07702ANDEWF119FI1000
   07780BLANJE220FO2100
   07850DALLJE119FI1050
   07668FEINPE220PL1950
   07972FIKETE210SE2500
   08001FLANJE119PL1920
   07715FLINGA119AD3000
   07961FREIHN220PL1780
   07643GREEJW119SE2700
   07650HALSPD210SE2000
   08100JODSWE210SE3150
   07760JONEKB119PL2200
   07883JONETW210SE2010
   07889KLEINM220PL1830
   07700LONDAJ220AD3000
   07970MANKCA119MA2410
   07671MARSJJ119FI1200
   07710MARTCH220PL1750
   07642MARTJT220PL1900
   07716MERLCH220FO2200
   07761REDFBB119SE2650
   07805ROPEES220PL2040
   07670SCHAWE119AD3100
   08000SCHEDR210FI2100
   07888WIENSH119MA2450
$
```

Example 4 Creating a Sequential File (Sheet 2 of 2)

Example 5

Example 5 illustrates how to sort a sequential file by the first five bytes of each record. The output file does not have the same attributes as the input file. This is because COBOL, rather than SORT, defines the file attributes. It would also be possible to define the file attributes with an output procedure to set the file attributes as desired using FDL$CREATE.

❶ This program sorts the file SEQ.DAT in ascending order by key ID_NUM. The output file is SORTED.DAT. The sort uses the standard GIVING clause, therefore input and output procedures are not used.

LEARNING ACTIVITY

(OPTIONAL) Refer to the *VAX COBOL User's Guide* for an explanation of the SORT statement phrases.

```
1       *                                                FILESORT.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. FILESORT.
5       *
6       *   This program illustrates the use of the COBOL SORT
7       *   verb to sort a file in ascending order by one key.
8       *
9       ENVIRONMENT DIVISION.
10      INPUT-OUTPUT SECTION.
11      FILE-CONTROL.
12
13          SELECT INPUT_FILE ASSIGN TO 'SEQ'
14            ORGANIZATION IS SEQUENTIAL.
15
16          SELECT OUTPUT_FILE ASSIGN TO 'SORTED'
17            ORGANIZATION IS SEQUENTIAL.
18
19          SELECT WORK_FILE ASSIGN TO SORT_WORK_FILE.
20
21      DATA DIVISION.
22      FILE SECTION.
23
24      FD  INPUT_FILE.
25      01  INPUT_RECORD            PIC X(20).
26
27      FD  OUTPUT_FILE.
28      01  OUTPUT_RECORD           PIC X(20).
29
30      SD  WORK_FILE.
31      01  WORK_RECORD.
32              02   ID_NUM                  PIC X(5).
33              02   EMP_NAME                PIC X(6).
34              02   EMP_DEPT                PIC XXX.
35              02   SKILL                   PIC XX.
36              02   SALARY                  PIC XXXX.
37
38      PROCEDURE DIVISION.
39      BEGIN.
40      *
41      *   Sort the file using ID_NUM as the key
42  (1)     SORT WORK_FILE ON ASCENDING KEY ID_NUM
43                  USING  INPUT_FILE
44                  GIVING OUTPUT_FILE.
45          STOP RUN.
```

Example 5 VAX COBOL Program Using VAX SORT to Perform File Sorting (Sheet 1 of 2)

```
$ COBOL FILESORT
$ LINK FILESORT
$ RUN FILESORT
$ TYPE SORTED.DAT
07642MARTJT220PL1900
07643GREEJW119SE2700
07650HALSPD210SE2000
07668FEINPE220PL1950
07670SCHAWE119AD3100
07671MARSJJ119FI1200
07672ALBEHA210SE2100
07700LONDAJ220AD3000
07702ANDEWF119FI1000
07710MARTCH220PL1750
07715FLINGA119AD3000
07716MERLCH220FO2200
07760JONEKB119PL2200
07761REDFBB119SE2650
07780BLANJE220FO2100
07805ROPEES220PL2040
07850DALLJE119FI1050
07883JONETW210SE2010
07888WIENSH119MA2450
07889KLEINM220PL1830
07961FREIHN220PL1780
07970MANKCA119MA2410
07972FIKETE210SE2500
08000SCHEDR210FI2100
08001FLANJE119PL1920
08100JODSWE210SE3150
$
```

Example 5 VAX COBOL Program Using VAX SORT to Perform File Sorting (Sheet 2 of 2)

Example 6

The record sorting facility allows you to pass individual records to SORT, then receive from SORT the same records in sorted order. Example 6 sorts a selected set of records from a file.

❶ You can manipulate data before and after sorting with the INPUT PROCEDURE and OUTPUT PROCEDURE phrases. The INPUT PROCEDURE allows you to manipulate data entering the sort. An INPUT PROCEDURE can be used to shorten data records, change the format of data records, or pass certain records to sort. The OUTPUT PROCEDURE allows you to maipulate the data returned from the sort.

❷ The INPUT PROCEDURE selects certain records to release to sort. Each record to be sorted is passed individually to a work file. When all the records are released, the input procedure completes and the records are sorted.

❸ After the sort completes, the program automatically transfers control to the OUTPUT PROCEDURE. The records are returned and displayed one at a time.

```
 1       *                                                RECSORT.COB
 2       IDENTIFICATION DIVISION.
 3       *
 4       PROGRAM-ID. RECSORT.
 5       *
 6       *   This program illustrates the use of the COBOL SORT
 7       *   to selectively sort the records from a file.
 8       *
 9       ENVIRONMENT DIVISION.
10       INPUT-OUTPUT SECTION.
11       FILE-CONTROL.
12
13           SELECT INPUT_FILE ASSIGN TO 'SEQ'
14             ORGANIZATION IS SEQUENTIAL.
15
16           SELECT WORK_FILE ASSIGN TO SORT_WORK_FILE.
17
18       DATA DIVISION.
19       FILE SECTION.
20
21       FD  INPUT_FILE.
22       01  INPUT_RECORD.
23              02   EMP_ID_NUM              PIC X(5).
24              02   FILLER                  PIC X(15).
25
26       SD  WORK_FILE.
27       01  WORK_RECORD.
28              02   ID_NUM                  PIC X(5).
29              02   EMP_NAME                PIC X(6).
30              02   EMP_DEPT                PIC XXX.
31              02   SKILL                   PIC XX.
32              02   SALARY                  PIC XXXX.
33
34       WORKING-STORAGE SECTION.
35       01  IOSTAT          PIC XX VALUE '00'.
36       01  SORT_DEPT       PIC X(5) VALUE '08000'.
37
38       PROCEDURE DIVISION.
39       BEGIN SECTION.
40       DO_SORT.
41       *
42       *   Sort the file using ID_NUM as the key
43           SORT WORK_FILE ON ASCENDING KEY ID_NUM
44       (1)         INPUT PROCEDURE IS TEST-DEPARTMENT
45                   OUTPUT PROCEDURE IS DISPLAY-RECORD.
46           STOP RUN.
47
48
```

Example 6 VAX COBOL Program Using VAX SORT to Sort a Group of Records (Sheet 1 of 2)

```
49  ❷   TEST-DEPARTMENT SECTION.
50      CHECK_EMP_DEPT.
51          OPEN INPUT INPUT_FILE.
52          PERFORM UNTIL IOSTAT IS NOT EQUAL '00'
53                  IF EMP_ID_NUM IS NOT LESS THAN SORT_DEPT
54                          THEN RELEASE WORK_RECORD FROM INPUT_RECORD
55                          END-IF
56          READ INPUT_FILE AT END MOVE '99' TO IOSTAT
57          END-READ
58          END-PERFORM.
59
60
61  ❸   DISPLAY-RECORD SECTION.
62      OUTPUT_RECORD.
63          RETURN WORK_FILE AT END STOP RUN.
64          DISPLAY WORK_RECORD.
65          GO TO OUTPUT_RECORD.
```

```
$ COBOL RECSORT
$ LINK RECSORT
$ RUN RECSORT
08000SCHEDR210FI2100
08001FLANJE119PL1920
08100JONDSWE210SE3150
$
```

Example 6 VAX COBOL Program Using VAX SORT to Sort a Group of Records (Sheet 2 of 2)

Lab Exercises

1. Modify Example 5 FILESORT.COB in this module so that it sorts the files SEQ1.DAT and SEQ2.DAT on the second field and in descending order. The records of SEQ1.DAT and SEQ2.DAT have the same format. A sample record follows:

```
          sort
          field
        |     |
07642MARTJT220PL1900
```

   Sort SEQ1.DAT and SEQ2.DAT.

   Refer to the *VAX COBOL User's Guide* for more information about the SORT verb.

2. Write a program that uses the COBOL MERGE verb to merge the sorted files created in Lab Exercise 1.

   Refer to the *VAX COBOL User's Guide* for more information about the MERGE verb.

3. Write a program that uses the COBOL SORT verb to sort the file LAB3.DAT. The program should:

   a. Use an input procedure that selects only input records with the RIG field = 'SLOOP' to be written to the output file.

   b. Use a GIVING clause for output.

Lab Solutions

1.

```
1       *                                                LABSOL1.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL1.
5       *
6       *   This program illustrates the use of the COBOL SORT
7       *   verb to sort a file in descending order by one key.
8       *
9       ENVIRONMENT DIVISION.
10      INPUT-OUTPUT SECTION.
11      FILE-CONTROL.
12
13          SELECT INPUT_FILE ASSIGN TO 'SEQ'
14            ORGANIZATION IS SEQUENTIAL.
15
16          SELECT OUTPUT_FILE ASSIGN TO 'SORTED'
17            ORGANIZATION IS SEQUENTIAL.
18
19          SELECT WORK_FILE ASSIGN TO SORT_WORK_FILE.
20
21      DATA DIVISION.
22      FILE SECTION.
23
24      FD  INPUT_FILE
25          VALUE OF ID IS INFILE.
26      01  INPUT_RECORD            PIC X(20).
27
28      FD  OUTPUT_FILE
29          VALUE OF ID IS OUTFILE.
30      01  OUTPUT_RECORD           PIC X(20).
31
32      SD  WORK_FILE.
33      01  WORK_RECORD.
34             02   ID_NUM                  PIC X(5).
35             02   EMP_NAME                PIC X(6).
36             02   EMP_DEPT                PIC XXX.
37             02   SKILL                   PIC XX.
38             02   SALARY                  PIC XXXX.
39
40      WORKING-STORAGE SECTION.
41
42      01  INFILE                  PIC X(30) VALUE SPACES.
43      01  OUTFILE                 PIC X(30) VALUE SPACES.
44      PROCEDURE DIVISION.
45      BEGIN.
46      *
47      *   Get input and output file names
48          DISPLAY 'Input file_name:  ' WITH NO ADVANCING.
49          ACCEPT INFILE.
50          DISPLAY 'Output file_name: ' WITH NO ADVANCING.
51          ACCEPT OUTFILE.
52      *
53      *   Sort the file using ID_NUM as the key
54          SORT WORK_FILE ON DESCENDING KEY EMP_NAME
55                  USING  INPUT_FILE
56                  GIVING OUTPUT_FILE.
57          STOP RUN.
```

2.

```
      *                                                  LABSOL2.COB
      IDENTIFICATION DIVISION.
      *
      PROGRAM-ID. LABSOL2.
      *
      *   This program illustrates the use of the COBOL MERGE
      *   verb to merge two identically sequenced files.
      *
      ENVIRONMENT DIVISION.
      INPUT-OUTPUT SECTION.
      FILE-CONTROL.

          SELECT INPUT_FILE_1 ASSIGN TO 'SORTED1'
            ORGANIZATION IS SEQUENTIAL.

          SELECT INPUT_FILE_2 ASSIGN TO 'SORTED2'
            ORGANIZATION IS SEQUENTIAL.

          SELECT OUTPUT_FILE ASSIGN TO 'MERGED'
            ORGANIZATION IS SEQUENTIAL.

          SELECT WORK_FILE ASSIGN TO MERGE_WORK_FILE.

      DATA DIVISION.
      FILE SECTION.

      FD  INPUT_FILE_1
          VALUE OF ID IS INFILE1.
      01  INPUT_RECORD_1          PIC X(20).

      FD  INPUT_FILE_2
          VALUE OF ID IS INFILE2.
      01  INPUT_RECORD_2          PIC X(20).

      FD  OUTPUT_FILE
          VALUE OF ID IS OUTFILE.
      01  OUTPUT_RECORD           PIC X(20).

      SD  WORK_FILE.
      01  WORK_RECORD.
             02   ID_NUM                PIC X(5).
             02   EMP_NAME              PIC X(6).
             02   EMP_DEPT              PIC XXX.
             02   SKILL                 PIC XX.
             02   SALARY                PIC XXXX.

      WORKING-STORAGE SECTION.

      01  INFILE1                 PIC X(30) VALUE SPACES.
      01  INFILE2                 PIC X(30) VALUE SPACES.
      01  OUTFILE                 PIC X(30) VALUE SPACES.
      PROCEDURE DIVISION.
      BEGIN.
      *
      *   Get input and output file names
          DISPLAY 'Input file_name_1:  ' WITH NO ADVANCING.
          ACCEPT INFILE1.
          DISPLAY 'Input file_name_2:  ' WITH NO ADVANCING.
          ACCEPT INFILE2.
          DISPLAY 'Output file_name: ' WITH NO ADVANCING.
          ACCEPT OUTFILE.
      *
      *   Merge the file using ID_NUM as the key
          MERGE WORK_FILE ON DESCENDING KEY EMP_NAME
                  USING  INPUT_FILE_1 INPUT_FILE_2
                  GIVING OUTPUT_FILE.
          STOP RUN.
```

3.

```
1       *                                                LABSOL3.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID.    LABSOL3.
5       *
6       *
7       *   This program sorts file LAB3.DAT. It uses an input
8       *           procedure and the standard GIVING clause for output.
9       *           The input procedure does a test on the input record
10      *           to see if the RIG type is SLOOP. If so, the record
11      *   is written to the output file; otherwise the record
12      *           is discarded.
13      *
14      ENVIRONMENT DIVISION.
15      INPUT-OUTPUT SECTION.
16      FILE-CONTROL.
17
18          SELECT IN-FILE ASSIGN TO 'LAB3.DAT'.
19          SELECT OUT-FILE ASSIGN TO 'LAB3.OUT'.
20          SELECT SORT-FILE ASSIGN TO 'SRTFIL'.
21
22      I-O-CONTROL.
23                  SAME RECORD AREA FOR IN-FILE SORT-FILE.
24      DATA DIVISION.
25
26      FILE SECTION.
27      SD  SORT-FILE.
28      01  SORT-REC.
29                  05      BUILDER.
30                          10      BUILD-1 PIC X(5).
31                          10      FILER   PIC X(5).
32                  05      MODEL   PIC X(10).
33                  05      RIG     PIC X(5).
34                  05      FILLER  PIC X.
35                  05      BOAT-LENGTH     PIC 99.
36                  05      FILLER  PIC X.
37                  05      WEIGHT  PIC 9(5).
38                  05      FILLER  PIC X.
39                  05      BEAM    PIC 99.
40                  05      FILLER  PIC X.
41                  05      PRICE   PIC 9(5).
42      FD  IN-FILE
43                  FILE STATUS IS FILE-STATUS.
44      01  IN-REC                 PIC X(43).
45      FD  OUT-FILE.
46      01  OUT-REC                PIC X(43).
47
48      WORKING-STORAGE SECTION.
49
50      01  FILE-STATUS     PIC XX  VALUE "00".
51      PROCEDURE DIVISION.
52      BEGIN SECTION.
53      DO-SORT.
54
55          SORT SORT-FILE ON ASCENDING KEY BEAM
56                          INPUT PROCEDURE IS TEST-RIG
57                  GIVING OUT-FILE
58                  DISPLAY 'END OF SORT.'.
59          STOP RUN.
60      TEST-RIG SECTION.
61      0.
62                  OPEN INPUT IN-FILE.
63                  PERFORM  UNTIL FILE-STATUS IS NOT EQUAL '00'
64                  IF RIG = 'SLOOP' THEN RELEASE SORT-REC
65                          END-IF
66                  READ IN-FILE AT END CONTINUE END-READ
67                  END-PERFORM.
```

Handling Exceptions and Exits

File Locations

On-line files are provided to help you master this module. The files are located in the directory:

```
DISK$COURSE:[COURSE.V4PROG.COB.HAND]
```

For your convenience, your system manager may have created the following logical name equivalence:

```
DISK$COURSE:[COURSE.V4PROG.COB.HAND] = V4PROG$COB$HAND
```

Three types of files are provided:

1. The Module Programming Examples, which are also listed in this language workbook.
2. Files that you modify to complete the Laboratory Exercises.
3. Solutions to the Laboratory Exercises.

If you wish to modify and/or compile, link, and run any of the programs, first copy the file into your directory.

Establishing a User-Written Condition Handler

Condition Handlers are procedures set up to handle hardware or software detected exceptions that interrupt the normal execution of a program. When a COBOL image is executed, four levels of condition handlers are established:

- COBOL handler
- The VAX/VMS default handlers:
  - Traceback handler
  - Catch-all handler
  - Last-chance handler

When any exception condition exists, the COBOL handler (COB$HANDLER) is automatically called. COB$HANDLER performs three functions:

1. Interacts with COB$IOEXCEPTION to perform a USE procedure
2. Resignals SS$__ROPERAND as COB$__INVDECDAT if the error is caused by a CVTTP (convert trailing numeric to packed) or DVTSP (convert leading separate to packed)
3. Resignals all other errors

It is possible for a COBOL program to establish its own handler to take the place of the COBOL established handler. However, it requires considerable knowledge of how condition handlers function.

When you establish a user-written condition handler in your main program, you replace the COBOL handler, which means that the two conditions tested for by COB$HANDLER will not be detected. The second of these, SS$__ROPERAND, would be detected by the traceback handler. However, if the first error occurred where a USE PROCEDURE issued an EXIT PROGRAM, unpredictable results would occur. It would probably be best not to establish a user-written handler with a USE PROCEDURE that causes an IMAGE EXIT.

If you must declare a condition handler in a COBOL program, it should not be in the main program. Instead, declare the condition handler in a subroutine. Then, when the condition handler is invoked, it should remedy the condition or resignal the error. If the error is resignaled, the system will call COB$HANDLER (the condition handler associated with the main program).

Example 1

This example shows a user-written condition handling routine that determines the reason for a system service failure. The example handler only handles one type of exception, system service failures. All other exceptions are resignaled, allowing them to be handled by the system default handlers.

This handler is useful because the system traceback handler only indicates that a system service failure has occurred, not which specific error caused the failure.

❶ The main program does not establish a user-written condition handler. The default handler, COB$HANDLER, is associated with the main program. A user-written condition handler is established in the subprogram.

❷ LIB$ESTABLISH is used by the subprogram to establish the condition handler SSHAND.

❸ System service failure mode is enabled so that errors in system service calls initiate a search for a condition handler.

❹ The condition handler is written as a function, with SS__HAND as the function result. SS__HAND is declared with a PIC S9(9) COMP to enable it to return a status code in R0.

❺ You must allocate space for the signal and mechanism arrays. Notice that the mechanism array always contains five elements, but the signal array varies according to the number of additional arguments.

❻ The handler checks the error condition to determine if it can be handled. The LIB$MATCH__COND routine would be useful here to check for one of a collection of conditions.

❼ The $GETMSG system service translates the error code into the associated error message. This system service, as well as the other capabilities of the message facility, are discussed in the Building Human Interfaces module.

❽ If the routine does not remedy the exception condition, it will return with a value of SS$__RESIGNAL.

❾ Output from user-written condition handling routine.

❿ Output from the system-defined condition handlers.

```
1       *                                                SSCOND.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. SSCOND.
5       *
6       *   This program defines and establishes its own
7       *   condition handling routine to handle system
8       *   service failure.
9       *
10      PROCEDURE DIVISION.
11      BEGIN.
12   (1)    CALL 'SSCOND_SUB'.
13          STOP RUN.
14      END PROGRAM SSCOND.
15
16
17      IDENTIFICATION DIVISION.
18      *
19      PROGRAM-ID. SSCOND_SUB.
20      DATA DIVISION.
21      WORKING-STORAGE SECTION.
22
23      01  SSHAND          PIC S9(9) COMP  VALUE EXTERNAL SSHAND.
24
25      PROCEDURE DIVISION.
26      BEGIN.
27      *
28      *   Establish condition handler
29   (2)    CALL 'LIB$ESTABLISH' USING BY VALUE SSHAND.
30      *
31      *   Enable system service failure mode
32   (3)    CALL 'SYS$SETSFM' USING BY VALUE 1.
33      *
34      *   Generate a bad system service call
35          CALL 'SYS$QIOW' USING BY VALUE 0 0 0 0
36                                         0 0 0 0
37                                         0 0 0 0.
38          STOP RUN.
39      END PROGRAM SSCOND_SUB.
40
41
42
43
44      IDENTIFICATION DIVISION.
45      *
46      PROGRAM-ID. SSHAND.
47      *
48      *   This routine is to be used as a condition handler
49      *   for system service failures.
50      *
51      DATA DIVISION.
52
53      WORKING-STORAGE SECTION.
54      01  SS_HAND         PIC S9(9) COMP.
55      01  SS$_SSFAIL      PIC S9(9) COMP  VALUE EXTERNAL SS$_SSFAIL.
56      01  SS$_RESIGNAL    PIC S9(9) COMP  VALUE EXTERNAL SS$_RESIGNAL.
57      01  MSGLEN          PIC S9(4) COMP.
58      01  MSGID           PIC S9(9) COMP.
59      01  ERRMSG          PIC X(80).
60      01  STAT            PIC S9(9) COMP.
61
```

Example 1 Using a Condition Handler (Sheet 1 of 2)

```
62        LINKAGE SECTION.
63        01  SIGNAL_ARRAY.
64            03      SIGNAL_ARGS     PIC 9(9) COMP.
65            03      SIGNAL          OCCURS 4 TO 10 TIMES
66        5                           DEPENDING ON SIGNAL_ARGS.
67                    05 SIGNAL_VALUE PIC S9(9) COMP.
68        01  MECHANISM_ARRAY.
69            03      MECH_ARGS       OCCURS 5 TIMES.
70                    05 MECH         PIC 9(9) COMP.
71
72        4   PROCEDURE DIVISION USING SIGNAL_ARRAY MECHANISM_ARRAY
73                        GIVING SS_HAND.
74        BEGIN.
75        6       IF SIGNAL_VALUE(1) NOT EQUAL SS$_SSFAIL THEN
76                    MOVE SS$_RESIGNAL TO SS_HAND
77                ELSE
78                    MOVE SIGNAL(2) TO MSGID
79        7           CALL 'SYS$GETMSG' USING BY VALUE MSGID
80                                            BY REFERENCE MSGLEN
81                                            BY DESCRIPTOR ERRMSG
82                                            BY VALUE 0 0
83                                        GIVING STAT
84                    IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
85                    DISPLAY 'System service call failed with error:'.
86                    DISPLAY ERRMSG(1:MSGLEN).
87        *
88        *           This is where the handler would perform
89        *           corrective measures
90        8       MOVE SS$_RESIGNAL TO SS_HAND.
91                EXIT PROGRAM.
92        END PROGRAM SSHAND.
```

```
$ COBOL SSCOND
$ LINK SSCOND
$ RUN SSCOND
System service call failed with error:
9  %SYSTEM-F-IVCHAN, invalid I/O channel
%SYSTEM-F-SSFAIL, system service failure exception, status=0000013C,
PC=7FFEDE06, PSL=03C00000
%TRACE-F-TRACEBACK, symbolic stack dump follows
module name     routine name            line       rel PC    abs PC
10
                                                   7FFEDE06  7FFEDE06
SSCOND_SUB      SSCOND_SUB                35       00000036  0000065A
SSCOND          SSCOND                    12       00000012  00000612
$
```

Example 1 Using a Condition Handler (Sheet 2 of 2)

Lab Exercises

1. Modify the program LAB1.COB to do the following:
   - Generate MTH$__FLOOVEMAT, MTH$__SQUROONEG and MTH$__LOGZERNEG errors.
   - Establish a condition handler that fixes MTH$__FLOOVEMAT and MTH$__SQUROONEG errors by assigning a value of zero as the function result; resignal all other errors.

Provided File

```
*                                              LAB1.COB
IDENTIFICATION DIVISION.
*
PROGRAM-ID.         LAB1.
*
*  This program calls a subroutine that generates
*  MTH$_FLOOVEMAT and MTH$_SQUROONEG errors. The
*  subroutine establishes a condition handler which
*  fixes a MTH$_FLOOVEMAT error (by replacing the
*  result with 0.0), and resignals all other errors.
*
PROCEDURE DIVISION.
BEGIN.
    CALL 'LAB1_SUB'.
    STOP RUN.
END PROGRAM LAB1.

IDENTIFICATION DIVISION.
PROGRAM-ID.         LAB1_SUB.
*
DATA DIVISION.
WORKING-STORAGE SECTION.

01 MTH_HANDLER      PIC S9(9) COMP VALUE EXTERNAL MTH_HANDLER.
01 X_EXP            COMP-1  VALUE 90.0.
01 Y_EXP            COMP-1  VALUE -1.0.
01 DISP_X           PIC Z(7)9.9.
01 DISP_Y           PIC Z(7)9.9.
```

```
PROCEDURE DIVISION.
BEGIN.
*
*  Establish condition handler
   CALL 'LIB$ESTABLISH' USING BY VALUE MTH_HANDLER.
*
*  Generate MTH$_FLOOVEMAT error
   CALL 'MTH$EXP' USING X_EXP GIVING X_EXP.
   MOVE X_EXP to DISP_X.
   DISPLAY 'Exponetial of X = ' DISP_X.
*
*  Generate MTH$_SQUROONEG error
   CALL 'MTH$SQRT' USING Y_EXP GIVING Y_EXP.
   MOVE Y_EXP TO DISP_Y.
   DISPLAY 'Square root of Y = ' DISP_Y.
   STOP RUN.
END PROGRAM LAB1_SUB.


IDENTIFICATION DIVISION.
*
PROGRAM-ID.          MTH_HANDLER.
*
DATA DIVISION.
WORKING-STORAGE SECTION.

01 MTH_HAND          PIC S9(9) COMP.
01 ERR_INDEX         PIC  9(9) COMP.
01 SS$_RESIGNAL      PIC S9(9) COMP VALUE EXTERNAL SS$_RESIGNAL.
01 SS$_CONTINUE      PIC S9(9) COMP VALUE EXTERNAL SS$_CONTINUE.
01 MTH$_FLOOVEMAT    PIC S9(9) COMP VALUE EXTERNAL MTH$_FLOOVEMAT.
01 MTH$_SQUROONEG    PIC S9(9) COMP VALUE EXTERNAL MTH$_SQUROONEG.

LINKAGE SECTION.

01 SIGNAL_ARRAY.
   03       SIGNAL_ARGS       PIC 9(9) COMP.
   03       SIGNAL            OCCURS 4 TO 10 TIMES
                              DEPENDING ON SIGNAL_ARGS.
            05 SIGNAL_VALUE   PIC 9(9) COMP.
01 MECHANISM_ARRAY.
   03       MECH_ARGS         OCCURS 5 TIMES.
            05 MECH           PIC 9(9) COMP.
```

(Sheet 2 of 3)

```
PROCEDURE DIVISION USING SIGNAL_ARRAY MECHANISM_ARRAY
                GIVING MTH_HAND.
BEGIN.
*
*  Check for handleable errors
   CALL 'LIB$MATCH_COND' USING BY REFERENCE SIGNAL(1)
                                       MTH$_FLOOVEMAT
                           GIVING ERR_INDEX.
   EVALUATE ERR_INDEX
           WHEN 0
*
*                  Resignal anything else
                   MOVE SS$_RESIGNAL TO MTH_HAND

           WHEN 1
*
*                  If MTH$_FLOOVEMAT error, fix it and continue
                   MOVE 0 TO MECH(4) MECH(5)
                   MOVE SS$_CONTINUE TO MTH_HAND
   END-EVALUATE.
   EXIT PROGRAM.
END PROGRAM MTH_HANDLER.
```

(Sheet 3 of 3)

2. Write a program that establishes a condition handler to print the message associated with a signaled condition. Have the program use LIB$SIGNAL to signal the condition SS$NORMAL.

   Refer to the *VAX/VMS Run-Time Library Reference Manual* for more information about Condition Handling.

3. Expand the condition handler you wrote for Lab Exercise 2 so that it outputs the contents of the signal array and the mechanism array.

Lab Solutions

1.

```
1       *                                                LABSOL1.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL1.
5       *
6       *   This program calls a subroutine that
7       *   establishes a condition handler which
8       *   fixes MTH$_FLOOVEMAT and MTH$_SQUROONEG errors
9       *   (by replacing the result with 0.0), and resignals
10      *   all other errors.
11      *
12      PROCEDURE DIVISION.
13      BEGIN.
14          CALL 'LABSOL1_SUB'.
15          STOP RUN.
16      END PROGRAM LABSOL1.
17
18
19      IDENTIFICATION DIVISION.
20      PROGRAM-ID. LABSOL1_SUB.
21      *
22      DATA DIVISION.
23      WORKING-STORAGE SECTION.
24
25      01  MTH_HANDLER     PIC S9(9) COMP  VALUE EXTERNAL MTH_HANDLER.
26      01  X_EXP           COMP-1  VALUE 90.0.
27      01  Y_EXP           COMP-1  VALUE -1.0.
28      01  Z_EXP           COMP-1  VALUE  0.0.
29      01  DISP_X          PIC Z(7)9.9.
30      01  DISP_Y          PIC Z(7)9.9.
31      01  DISP_Z          PIC Z(7)9.9.
32
33      PROCEDURE DIVISION.
34      BEGIN.
35      *
36      *   Establish condition handler
37          CALL 'LIB$ESTABLISH' USING BY VALUE MTH_HANDLER.
38      *
39      *   Generate MTH$_FLOOVEMAT error
40          CALL 'MTH$EXP' USING X_EXP  GIVING X_EXP.
41          MOVE X_EXP to DISP_X.
42          DISPLAY 'Exponential of X = ' DISP_X.
43      *
44      *   Generate MTH$_SQUROONEG error
45          CALL 'MTH$SQRT' USING Y_EXP  GIVING Y_EXP.
46          MOVE Y_EXP TO DISP_Y.
47          DISPLAY 'Square root of Y = ' DISP_Y.
48      *
49      *   Generate MTH$_LOGZERNEG error
50          CALL 'MTH$ALOG' USING Z_EXP  GIVING Z_EXP.
51          MOVE Z_EXP TO DISP_Z.
52          DISPLAY 'Log of Z = ' DISP_Z.
53          STOP RUN.
54      END PROGRAM LABSOL1_SUB.
55
56
57
```

```
58      IDENTIFICATION DIVISION.
59      *
60      PROGRAM-ID. MTH_HANDLER.
61      *
62      DATA DIVISION.
63      WORKING-STORAGE SECTION.
64
65      01  MTH_HAND         PIC S9(9) COMP.
66      01  ERR_INDEX        PIC  9(9) COMP.
67      01  SS$_RESIGNAL     PIC S9(9) COMP VALUE EXTERNAL SS$_RESIGNAL.
68      01  SS$_CONTINUE     PIC S9(9) COMP VALUE EXTERNAL SS$_CONTINUE.
69      01  MTH$_FLOOVEMAT   PIC S9(9) COMP VALUE EXTERNAL MTH$_FLOOVEMAT.
70      01  MTH$_SQUROONEG   PIC S9(9) COMP VALUE EXTERNAL MTH$_SQUROONEG.
71      01  MTH$_LOGZERNEG   PIC S9(9) COMP VALUE EXTERNAL MTH$_LOGZERNEG.
72
73      LINKAGE SECTION.
74
75      01  SIGNAL_ARRAY.
76          03      SIGNAL_ARGS     PIC 9(9) COMP.
77          03      SIGNAL          OCCURS 4 TO 10 TIMES
78                                  DEPENDING ON SIGNAL_ARGS.
79                  05 SIGNAL_VALUE PIC 9(9) COMP.
80      01  MECHANISM_ARRAY.
81          03      MECH_ARGS       OCCURS 5 TIMES.
82                  05 MECH         PIC 9(9) COMP.
83
84      PROCEDURE DIVISION USING SIGNAL_ARRAY MECHANISM_ARRAY
85                       GIVING MTH_HAND.
86      BEGIN.
87      *
88      *   Check for handleable errors
89          CALL 'LIB$MATCH_COND' USING BY REFERENCE SIGNAL(1)
90                                                   MTH$_SQUROONEG
91                                                   MTH$_FLOOVEMAT
92                                  GIVING ERR_INDEX.
93          EVALUATE ERR_INDEX
94                  WHEN 0
95      *
96      *                   Resignal anything else
97                          MOVE SS$_RESIGNAL TO MTH_HAND
98
99                  WHEN 1
100     *
101     *                   If MTH$SQUROONEG error, print
102     *                   warning and continue
103                         DISPLAY 'Warning - Sq Root of Neg Num'
104                         MOVE 0 TO MECH(4) MECH(5)
105                         MOVE SS$_CONTINUE TO MTH_HAND
106                 WHEN 2
107     *
108     *                   If MTH$_FLOOVEMAT error, fix it and continue
109                         MOVE 0 TO MECH(4) MECH(5)
110                         MOVE SS$_CONTINUE TO MTH_HAND
111         END-EVALUATE.
112         EXIT PROGRAM.
113     END PROGRAM MTH_HANDLER.
```

(Sheet 2 of 2)

2.

```
1       *                                                LABSOL2.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL2.
5       *
6       *   This program calls a subroutine that
7       *   specifies a condition handler which
8       *   formats and prints the error message(s)
9       *   associated with the signaled condition.
10      *
11      PROCEDURE DIVISION.
12      BEGIN.
13          CALL 'LAB2_SUB'.
14          STOP RUN.
15      END PROGRAM LABSOL2.
16
17
18      IDENTIFICATION DIVISION.
19      PROGRAM-ID. LAB2_SUB.
20      *
21      DATA DIVISION.
22      WORKING-STORAGE SECTION.
23
24      01  COND_HANDLER    PIC S9(9) COMP  VALUE EXTERNAL COND_HANDLER.
25      01  SS$_NORMAL      PIC S9(9) COMP  VALUE EXTERNAL SS$_NORMAL.
26
27      PROCEDURE DIVISION.
28      BEGIN.
29      *
30      *   Establish the condition handler
31          CALL 'LIB$ESTABLISH' USING BY VALUE COND_HANDLER.
32      *
33      *   Signal a condition
34          CALL 'LIB$SIGNAL' USING BY VALUE SS$_NORMAL.
35      END PROGRAM LAB2_SUB.
36
```

(Sheet 1 of 2)

```
37
38      IDENTIFICATION DIVISION.
39      *
40      PROGRAM-ID. COND_HANDLER.
41      *
42      DATA DIVISION.
43      WORKING-STORAGE SECTION.
44
45      01  COND_HAND        PIC S9(9) COMP.
46      01  SS$_CONTINUE     PIC S9(9) COMP VALUE EXTERNAL SS$_CONTINUE.
47      01  STAT             PIC S9(9) COMP.
48
49      LINKAGE SECTION.
50
51      01  SIGNAL_ARRAY.
52          03      SIGNAL_ARGS     PIC 9(9) COMP.
53          03      SIGNAL          OCCURS 4 TO 10 TIMES
54                                  DEPENDING ON SIGNAL_ARGS.
55                  05 SIGNAL_VALUE PIC 9(9) COMP.
56      01  MECHANISM_ARRAY.
57          03      MECH_ARGS       OCCURS 5 TIMES.
58                  05 MECH         PIC 9(9) COMP.
59
60      PROCEDURE DIVISION USING SIGNAL_ARRAY MECHANISM_ARRAY
61                       GIVING COND_HAND.
62      BEGIN.
63      *
64      *   Format/Print the error message(s)
65          COMPUTE SIGNAL_ARGS = SIGNAL_ARGS - 2
66          CALL 'SYS$PUTMSG' USING BY REFERENCE SIGNAL_ARRAY
67                                  BY VALUE 0 0
68                                  GIVING STAT.
69          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
70          MOVE SS$_CONTINUE TO COND_HAND.
71          EXIT PROGRAM.
72      END PROGRAM COND_HANDLER.
```

(Sheet 2 of 2)

3.

```
1       *                                                LABSOL3.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL3.
5       *
6       *   This program calls a subroutine to establish a
7       *   condition handler.  The condition handler:
8       *      1. Prints the contents of the mechanism array
9       *      2. Prints the contents of the signal array
10      *      3. Formats and prints the error message(s)
11      *         associated with the signaled condition.
12      *
13      PROCEDURE DIVISION.
14      BEGIN.
15          CALL 'LAB3_SUB'.
16          STOP RUN.
17      END PROGRAM LABSOL3.
18
19
20
21
22      IDENTIFICATION DIVISION.
23      PROGRAM-ID. LAB3_SUB.
24      *
25      DATA DIVISION.
26      WORKING-STORAGE SECTION.
27
28      01  COND_HANDLER    PIC S9(9) COMP VALUE EXTERNAL COND_HANDLER.
29      01  SS$_NORMAL      PIC S9(9) COMP VALUE EXTERNAL SS$_NORMAL.
30
31      PROCEDURE DIVISION.
32      BEGIN.
33      *
34      *   Establish condition handler
35          CALL 'LIB$ESTABLISH' USING BY VALUE COND_HANDLER.
36      *
37      *   Signal a condition
38          CALL 'LIB$SIGNAL' USING BY VALUE SS$_NORMAL.
39          STOP RUN.
40      END PROGRAM LAB3_SUB.
41
42
43
44      IDENTIFICATION DIVISION.
45      *
46      PROGRAM-ID. COND_HANDLER.
47      *
48      DATA DIVISION.
49      WORKING-STORAGE SECTION.
50
51      01  SS_HAND         PIC S9(9) COMP.
52      01  SS$_CONTINUE    PIC S9(9) COMP  VALUE EXTERNAL SS$_CONTINUE.
53      01  ARGUMENT                PIC S9(9) COMP.
54      01  DISPLAY_ARGUMENT        PIC Z(8)9.
55      01  STAT                    PIC S9(9) COMP.
56      01  I                       PIC S9(9) COMP.
57
```

(Sheet 1 of 2)

```
58        LINKAGE SECTION.
59
60        01  SIGNAL_ARRAY.
61            03      SIGNAL_ARGS     PIC 9(9) COMP.
62            03      SIGNAL          OCCURS 4 TO 10 TIMES
63                                    DEPENDING ON SIGNAL_ARGS.
64                    05 SIGNAL_VALUE PIC 9(9) COMP.
65        01  MECHANISM_ARRAY.
66            03      MECH_ARGS       OCCURS 5 TIMES.
67                    05 MECH         PIC 9(9) COMP.
68
69        PROCEDURE DIVISION USING SIGNAL_ARRAY MECHANISM_ARRAY
70                         GIVING SS_HAND.
71        BEGIN.
72        *
73        *   Print the signal array
74            DISPLAY ' '.
75            DISPLAY '    SIGNAL ARRAY'.
76            MOVE SIGNAL_ARGS TO DISPLAY_ARGUMENT..
77            DISPLAY 'Number of arguments ' DISPLAY_ARGUMENT.
78            MOVE SIGNAL(1) TO ARGUMENT.
79            PERFORM CONVERT.
80            DISPLAY 'Condition name:     ' DISPLAY_ARGUMENT.
81            PERFORM WITH TEST BEFORE VARYING I FROM 1 BY 1
82            UNTIL I > SIGNAL_ARGS - 3
83                    MOVE SIGNAL(I) TO ARGUMENT
84                    PERFORM CONVERT
85                    DISPLAY '                    ' DISPLAY_ARGUMENT
86            END-PERFORM
87            SUBTRACT 1 FROM SIGNAL_ARGS GIVING I.
88            MOVE SIGNAL(I) TO ARGUMENT.
89            PERFORM CONVERT.
90            DISPLAY 'PC:                 ' DISPLAY_ARGUMENT.
91            MOVE SIGNAL(SIGNAL_ARGS) TO ARGUMENT.
92            PERFORM CONVERT.
93            DISPLAY 'PSL:                ' DISPLAY_ARGUMENT.
94        *
95        *   Print the mechanism array
96            DISPLAY ' '.
97            DISPLAY '    MECHANISM ARRAY'.
98            MOVE MECH(2) TO ARGUMENT.
99            PERFORM CONVERT.
100           DISPLAY 'Establisher frame:  ' DISPLAY_ARGUMENT.
101           MOVE MECH(3) TO ARGUMENT.
102           PERFORM CONVERT.
103           DISPLAY 'Depth:              ' DISPLAY_ARGUMENT.
104           MOVE MECH(4) TO ARGUMENT.
105           PERFORM CONVERT.
106               DISPLAY 'R0:                 ' DISPLAY_ARGUMENT.
107           MOVE MECH(5) TO ARGUMENT.
108           PERFORM CONVERT.
109           DISPLAY 'R1:                 ' DISPLAY_ARGUMENT.
110       *
111       *   Format/print the error message(s)
112               COMPUTE SIGNAL_ARGS = SIGNAL_ARGS - 2
113           DISPLAY ' '.
114           CALL 'SYS$PUTMSG' USING SIGNAL_ARRAY
115                                   BY VALUE 0 0.
116           MOVE SS$_CONTINUE TO SS_HAND.
117           EXIT PROGRAM.
118
119       CONVERT.
120       *
121       *   Convert arguments to hexidecimal
122           CALL 'OTS$CVT_L_TZ' USING BY REFERENCE ARGUMENT
123                                     BY DESCRIPTOR DISPLAY_ARGUMENT
124                                     BY VALUE 8 4
125                               GIVING STAT.
126
127       END PROGRAM COND_HANDLER.
```

(Sheet 2 of 2)

Sharing Code and Data

File Locations

On-line files are provided to help you master this module. The files are located in the directory:

```
DISK$COURSE:[COURSE.V4PROG.COB.SHAR]
```

For your convenience, your system manager may have created the following logical name equivalence:

```
DISK$COURSE:[COURSE.V4PROG.COB.SHAR] = V4PROG$COB$SHAR
```

Three types of files are provided:

1. The Module Programming Examples, which are also listed in this language workbook.
2. Files that you modify to complete the Laboratory Exercises.
3. Solutions to the Laboratory Exercises.

If you wish to modify and/or compile, link, and run any of the programs, first copy the file into your directory.

Example 1

This example uses an installed shareable image to share data between two processes.

❶ The shared data area is declared as DATA__ARRAY in the source code for the shareable image.

❷ The shared data area is declared as DATA__ARRAY, using the same name in each program and in the shareable image.

❸ The program named SHDATA1 writes numbers to the data area. Note that no open statement or special addressing is needed to access the shared data area.

❹ The program named SHDATA2 reads the data in the shared area and outputs it to the terminal.

❺ COBOL does not generate position-independent references to DATA__ARRAY. Therefore, the linker cannot create a position-independent shareable image. The linker selects an address to base the shareable image when the SHARDATA is linked with SHDATA2. This base address is incompatible for SHDATA2. The BASE=n option on the LINK command manually bases the image at an address compatible for both SHDAT1 and SHDATA2.

In general, using the BASE=n option to create based images is not recommended. VAX/VMS memory manangement cannot relocate based images in the virtual address space; this may result in fragmentation of the virtual address space.

❻ The shareable image is copied to SYS$SHARE to be installed (copying to a system directory requires SYSPRV). Alternately, the image can be installed using the full file specification, or by assigning a logical name for the full file name.

❼ Each program that accesses the shared data area must be linked with the shareable image.

```
1       *                                                SHARDATA.COB
 2       IDENTIFICATION DIVISION.
 3       *
 4       PROGRAM-ID. SHARDATA.
 5       *
 6       *   This is a blockdata program which is linked and
 7       *   installed as a writeable shareable image.  It is
 8       *   shared by SHDATA1.COB and SHDATA2.COB
 9       *
10       DATA DIVISION.
11       WORKING-STORAGE SECTION.
12    (1)01  DATA_ARRAY IS EXTERNAL.
13          02       DIM1            OCCURS  3 TIMES.
14          03       DIM2            OCCURS 50 TIMES.
15          04       IARRAY          PIC 999 COMP.


 1       *                                                SHDATA1.COB
 2       IDENTIFICATION DIVISION.
 3       *
 4       PROGRAM-ID. SHDATA1.
 5       *
 6       *   This program wites to a shared data area in the
 7       *   installed shareable image called SHARDATA.  The
 8       *   program must be linked with SHARDATA in order to
 9       *   execute.
10       *
11       DATA DIVISION.
12       WORKING-STORAGE SECTION.
13    (2)01  DATA_ARRAY IS EXTERNAL.
14          02       DIM1                    OCCURS  3 TIMES.
15          03       DIM2                    OCCURS 50 TIMES.
16          04       IARRAY          PIC 999 COMP.
17       01  INUMBER          PIC 999 COMP    VALUE 0.
18       01  OUT_IARRAY       PIC Z(5)9.
19       01  I                PIC 999.
20       01  J                PIC 999.
21       PROCEDURE DIVISION.
22       BEGIN.
23           PERFORM VARYING I FROM 1 BY 1 UNTIL I > 3
24              PERFORM VARYING J FROM 1 BY 1 UNTIL J > 10
25    (3)           COMPUTE INUMBER = INUMBER + 5
26                  MOVE INUMBER TO IARRAY(I,J)
27              END-PERFORM
28           END-PERFORM
29           DISPLAY 'Write to global data area completed.'
30           STOP RUN.
```

Example 1 Sharing Data Through an Installed Shareable Image (Sheet 1 of 3)

```
1       *                                                SHDATA2.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. SHDATA2.
5       *
6       *   This program reads the data stored in the installed
7       *   shareable image called SHARDATA.  It must be linked
8       *   with the shareable image to execute properly.
9       *
10      DATA DIVISION.
11      WORKING-STORAGE SECTION.
12
13   ❷  01  DATA_ARRAY IS EXTERNAL.
14         02       DIM1                    OCCURS  3 TIMES.
15         03       DIM2                    OCCURS 50 TIMES.
16         04       IARRAY          PIC 999 COMP.
17      01  I               PIC 99 COMP.
18      01  J               PIC 99 COMP.
19      01  DISP_J          PIC Z9.
20      01  DUMMY.
21         02       OUT_IARRAY
22                  PIC Z(3)9       OCCURS 3 TIMES.
23
24      PROCEDURE DIVISION.
25      BEGIN.
26          DISPLAY '          Contents of global section'.
27          DISPLAY ' Row            1        2        3'.
28          DISPLAY ' Column'.
29          PERFORM VARYING J FROM 1 BY 1 UNTIL J > 10
30             MOVE J TO DISP_J
31             PERFORM VARYING I FROM 1 BY 1 UNTIL I > 3
32                  MOVE IARRAY(I,J) TO OUT_IARRAY(I)
33          END-PERFORM
34   ❹      DISPLAY ' ' DISP_J
35                 '          ' OUT_IARRAY(1)
36                 '      ' OUT_IARRAY(2)
37                 '      ' OUT_IARRAY(3)
38          END-PERFORM
39          STOP RUN.
```

Example 1 Sharing Data Through an Installed Shareable Image (Sheet 2 of 3)

```
$ COBOL SHARDATA
$ LINK/SHARE SHARDATA,SYS$INPUT/OPTIONS
❺ BASE=3000
*EXIT*
$ SET PROCESS/PRIVILEGE=SYSPRV
❻ $ COPY SHARDATA.EXE SYS$SHARE:SHARDATA.EXE
$ SET PROCESS/PRIVILEGE=NOSYSPRV
$ COBOL SHDATA1, SHDATA2
$ LINK SHDATA1, SYS$INPUT/OPTIONS
SYS$SHARE:SHARDATA/SHARE
❼ *EXIT*
$ LINK SHDATA2, SYS$INPUT/OPTIONS
SYS$SHARE:SHARDATA/SHARE
$EXIT*
$ SET PROCESS/PRIVILEGE=CMKRNL
$ RUN SYS$SYSTEM:INSTALL
INSTALL> SYS$SHARE:SHARDATA/WRITE/SHARE
INSTALL> /EXIT
$ SET PROCESS/PRIVILEGE=NOCMKRNL
$
$ RUN SHDATA1
Write to global data area completed.
$ RUN SHDATA2
          Contents of global section
 Row            1        2        3
 Column
  1             5       55      105
  2            10       60      110
  3            15       65      115
  4            20       70      120
  5            25       75      125
  6            30       80      130
  7            35       85      135
  8            40       90      140
  9            45       95      145
 10            50      100      150
$
```

Example 1 Sharing Data Through an Installed Shareable Image (Sheet 3 of 3)

Example 2

The program called SHAREDFIL is used to update records in a relative file. The ALLOWING clause is specified on the OPEN statement to invoke the RMS file sharing facility. In this example, the same program is used to access the file from two processes.

❶ This program requires a relative file named REL.DAT.

❷ These statements define the symbolic status codes returned by COBOL I/O statements.

❸ This section checks for errors when accessing the relative file.

❹ The ALLOWING clause is used on the OPEN statement to indicate that the file can be shared. Since manual locking was not specified, RMS automatically controls access to the file. Only read and update operations are allowed in this example. No new records may be written to the file.

❺ When accessing a relative file, COBOL does not check for maximum record number. You can check for the maximum number of records in your program as shown here.

❻ The second process is not allowed to access record number 2 while the first process is accessing it.

❼ Once the first process has finished with record number 2, the second process can update it.

LEARNING ACTIVITY

Refer to the *VAX COBOL User's Guide* for more information on file sharing and record locking for sequential, relative, and indexed files.

```
1       *                                                SHAREDFIL.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. SHAREDFIL.
5       *
6       *   This program can be run from two or more processes
7       *   to illustrate the use of an RMS shared file to share
8       *   data.  The program requires the existence of a
9       *   relative file named REL.DAT.
10      *
11      ENVIRONMENT DIVISION.
12      INPUT-OUTPUT SECTION.
13      FILE-CONTROL.
14
15  (1) SELECT RELATIVE_FILE
16          ASSIGN TO 'REL.DAT'
17          ORGANIZATION IS RELATIVE
18          FILE STATUS IS IOSTAT.
19
20      SELECT TERMNAL
21          ASSIGN TO 'SYS$INPUT'
22          ORGANIZATION IS SEQUENTIAL
23          FILE STATUS IS IOSTAT.
24
25      DATA DIVISION.
26
27      FILE SECTION.
28      FD  RELATIVE_FILE
29          ACCESS MODE IS RANDOM
30          RELATIVE KEY IS REC_NUM
31          RECORD VARYING DEPENDING ON REC_LEN.
32      01  RECERD              PIC X(20).
33
34      FD  TERMNAL
35          RECORD VARYING DEPENDING ON REC_LEN
36          ACCESS MODE IS SEQUENTIAL.
37      01  NEW_RECORD          PIC X(20).
38
39      WORKING-STORAGE SECTION.
40      01  REC_NUM             PIC 999.
41      01  REC_LEN             PIC  9(9) COMP.
42  (2) 01  IOSTAT              PIC XX.
43          88 SUCCESSFUL       VALUE '00'.
44          88 LOCKED           VALUE '92'.
45          88 OUT_RANGE        VALUE '23'.
46
47      PROCEDURE DIVISION.
48      DECLARATIVES.
49  (3) I-O-ERROR SECTION.
50          USE AFTER STANDARD ERROR PROCEDURE ON RELATIVE_FILE.
51      CHECK-ERROR-CODES.
52          IF OUT_RANGE DISPLAY 'Nonexistant record.'
53                  ELSE IF LOCKED DISPLAY 'Record locked by someone else.'
54                          ELSE CALL 'LIB$SIGNAL' USING
55                          BY VALUE RMS-STS OF RELATIVE_FILE
56                                   RMS-STV OF RELATIVE_FILE
57                  END-IF
58          END-IF.
59      END DECLARATIVES.
```

Example 2 Using an RMS Shared File (Sheet 1 of 2)

```
60          *
61          RECORD-ACCESS SECTION.
62          BEGIN.
63          *
64          *   Get records by record number until e-o-f
65          *           Prompt for record number
66      4       OPEN I-O RELATIVE_FILE ALLOWING UPDATERS.
67              OPEN INPUT TERMNAL.
68
69          GET-NEXT-RECORD.
70              DISPLAY 'Record number (CTRL Z to quit): ' WITH NO ADVANCING.
71              ACCEPT REC_NUM AT END STOP RUN.
72          *
73          *           Read record by record number
74      5       IF REC_NUM GREATER THAN 10
75                      DISPLAY 'Record number out of range.'
76              ELSE
77                      READ RELATIVE_FILE
78                      IF SUCCESSFUL PERFORM REPLACE-RECORD.
79              GO TO GET-NEXT-RECORD.
80
81          REPLACE-RECORD.
82              DISPLAY RECERD(1:REC_LEN).
83              INITIALIZE RECERD.
84          *
85          *   Request updated record
86              DISPLAY 'New value or CR: ' WITH NO ADVANCING.
87              READ TERMNAL AT END MOVE 0 TO REC_LEN.
88              IF REC_LEN NOT EQUAL 0 THEN
89                      REWRITE RECERD FROM NEW_RECORD(1:REC_LEN).
```

```
$ COBOL SHAREDFIL
$ LINK SHAREDFIL
$ RUN SHAREDFIL
Record number (CTRL Z to quit): 002
MSPIGGY
New value or CR: FOZZIE
Record number (CTRL Z to quit): 001
KERMIT
New value or CR:
Record number (CTRL Z to quit): ^Z
$
```

```
$ RUN SHAREDFIL
Record number (CTRL Z to quit): 002
6 Record locked by someone else.
Record number (CTRL Z to quit): 002
Record locked by someone else.
Record number (CTRL Z to quit): 002
7 FOZZIE
New value or CR: MSPIGGY
Record number (CTRL Z to quit): ^Z
$
```

Example 2 Using an RMS Shared File (Sheet 2 of 2)

Lab Exercises

1. Create an object module library containing LAB1A.OBJ and LAB1B.OBJ. LAB1A.COB and LAB1B.COB are listed below. Write a program that calls both modules in the newly created library.

Provided File

```
*                                                LAB1A.COB
IDENTIFICATION DIVISION.
*
PROGRAM-ID.          LAB1A.
*
* This procedure:
*  1. prompts for an integer
*  2. Multiply the integer by 3
*  3. Continues to prompt for an integer unitl on is
*     entered which is equal to the number in step 2.
*
DATA DIVISION.
WORKING-STORAGE SECTION.

01 NUM_X_3           PIC 999.
01 NEW_NUM           PIC 999.

PROCEDURE DIVISION.
BEGIN.
*
* Get first integer
  DISPLAY 'Enter an integer: ' WITH NO ADVANCING.
  ACCEPT NUM_X_3.
*
* Multiply integer by 3
  MULTIPLY 3 BY NUM_X_3.
*
* get next integer
GUESS_NUMBER.
  DISPLAY 'Enter an integer:  ' WITH NO ADVANCING.
  ACCEPT NEW_NUM
*
* Compare integers
  IF NEW_NUM EQUAL NUM_X_3 THEN
* Print correct/incorrect message
          DISPLAY 'The number is correct.'
  ELSE
          DISPLAY 'The number is incorrect.'
          GO TO GUESS_NUMBER.
  EXIT PROGRAM.
END PROGRAM LAB1A.
```

```
*                                          LAB1A.COB
IDENTIFICATION DIVISION.
*
PROGRAM-ID.          LAB1A.
*
* This procedure:
*  1. Prompts for a seed
*  2. Generates and prints 10 random numbers
*
DATA DIVISION.
WORKING-STORAGE SECTION.

01 INPUT_SEED      PIC 99.
01 SEED            PIC 9(9) COMP.
01 RANDOM_NUMBER   COMP-1.
01 OUT_RAN_NUM     PIC 9.9(8)

PROCEDURE DIVISION.
BEGIN.
*
* Get seed
  DISPLAY 'Enter a seed integer: ' WITH NO ADVANCING.
  ACCEPT SEED.
*
* Get and print 10 random numbers
PERFORM 10 TIMES
        CALL 'MTH$RANDOM' USING SEED GIVING RANDOM_NUMBER
        MOVE RANDOM_NUMBER TO OUT_RAN_NUM
        DISPLAY OUT_RAN_NUM
  END-PERFORM.
  EXIT PROGRAM.
END PROGRAM LAB1B.
```

(Sheet 2 of 2)

2. Create a shareable image library using the modules LAB1A.EXE and LAB1B.EXE created from LAB1A.COB and LAB1B.COB from Lab Exercise 1. Access the shareable image library using the program you wrote for Lab Exercise 1.

3. Write programs similar to SHARDATA.COB, SHARDATA1.COB, and SHARDATA2.COB (in this module) that will do the following:

   a. Write 50 random numbers into each of two arrays in a common data area.

   b. Output the average (mean) of the data items in each of the arrays.

   c. Set up the common data area. This must be installed as a shareable image. Remember that installing an image requires special privileges.

Lab Solutions

1.
```
1       *                                               LABSOL1.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL1.
5       *
6       *   This procedure calls LAB1A and LAB1B in a object
7       *   library.
8       *
9       PROCEDURE DIVISION.
10      BEGIN.
11          CALL 'LAB1A'.
12          CALL 'LAB1B'.
13          STOP RUN.
```

```
$ COBOL LAB1A,LAB1B,LABSOL1
$ LIBRARY/CREATE MYLIB LAB1A,LAB1B
$ LINK LABSOL1,MYLIB/LIBRARY
$ RUN LABSOL1
```

2.

```
 1      *                                                LABSOL1.COB
 2      IDENTIFICATION DIVISION.
 3      *
 4      PROGRAM-ID. LABSOL1.
 5      *
 6      *   This procedure calls LAB1A and LAB1B in a object
 7      *   library.
 8      *
 9      PROCEDURE DIVISION.
10      BEGIN.
11          CALL 'LAB1A'.
12          CALL 'LAB1B'.
13          STOP RUN.


 1      *                                                LABSOL2A.COB
 2      IDENTIFICATION DIVISION.
 3      *
 4      PROGRAM-ID. LABSOL2A.
 5      *
 6      *   This procedure calls LAB1A and LAB1B in a object
 7      *   library.
 8      *
 9      PROCEDURE DIVISION.
10      BEGIN.
11          CALL 'LAB1A'.
12          CALL 'LAB1B'.
13          STOP RUN.
14      END PROGRAM LABSOL2A.
15
16
17      *                                                LAB1A.COB
18      IDENTIFICATION DIVISION.
19      *
20      PROGRAM-ID. LAB1A.
21      *
22      *   This procedure:
23      *     1. prompts for an integer
24      *     2. Multiply the integer by 3
25      *     3. Continues to prompt for an integer unitl on is
26      *        entered which is equal to the number in step 2.
27      *
28      DATA DIVISION.
29      WORKING-STORAGE SECTION.
30
31      01  NUM_X_3          PIC 999.
32      01  NEW_NUM          PIC 999.
33
```

(Sheet 1 of 3)

```
34        PROCEDURE DIVISION.
35        BEGIN.
36        *
37        *    Get first integer
38             DISPLAY 'Enter an integer: ' WITH NO ADVANCING.
39             ACCEPT NUM_X_3.
40        *
41        *    Multiply integer by 3
42             MULTIPLY 3 BY NUM_X_3.
43        *
44        *    get next integer
45        GUESS_NUMBER.
46             DISPLAY 'Enter an integer: ' WITH NO ADVANCING.
47             ACCEPT NEW_NUM
48        *
49        *    Compare integers
50             IF NEW_NUM EQUAL NUM_X_3 THEN
51        *    Print correct/incorrect message
52                     DISPLAY 'The number is correct.'
53             ELSE
54                     DISPLAY 'The number is incorrect.'
55                     GO TO GUESS_NUMBER.
56             EXIT PROGRAM.
57        END PROGRAM LAB1A.
58
59
60        *                                                    LAB1B.COB
61         IDENTIFICATION DIVISION.
62         *
63         PROGRAM-ID. LAB1B.
64         *
65         *    This procedure:
66         *     1. Prompts for a seed
67         *     2. Generates and prints 10 random numbers
68         *
69         DATA DIVISION.
70         WORKING-STORAGE SECTION.
71
72         01  INPUT_SEED       PIC 99.
73         01  SEED             PIC 9(9) COMP.
74         01  RANDOM_NUMBER    COMP-1.
75         01  OUT_RAN_NUM      PIC 9.9(8).
76
```

(Sheet 2 of 3)

```
77        PROCEDURE DIVISION.
78        BEGIN.
79        *
80        *   Get seed
81            DISPLAY 'Enter a seed integer: ' WITH NO ADVANCING.
82            ACCEPT SEED.
83        *
84        *   Get and print 10 random numbers
85            PERFORM 10 TIMES
86                    CALL 'MTH$RANDOM' USING SEED GIVING RANDOM_NUMBER
87                    MOVE RANDOM_NUMBER TO OUT_RAN_NUM
88                    DISPLAY OUT_RAN_NUM
89            END-PERFORM.
90            EXIT PROGRAM.
91        END PROGRAM LAB1B.
```

```
 1        ;                                                  LABSOL2B.MAR
 2            .TITLE  LABSOL2B
 3        ;
 4        ;   This is the transfer vector file for the shareable
 5        ;   library procedures for LAB2.
 6        ;
 7            .PSECT  TRANSFER_VECTOR PIC, SHR, NOWRT, QUAD
 8            .TRANSFER       LAB1A
 9            .MASK           LAB1A
10            BRW             LAB1A+2
11            RET
12
13            .ALIGN  QUAD
14            .TRANSFER       LAB1B
15            .MASK           LAB1B
16            BRW             LAB1B+2
17            RET
18
19            .END
```

```
 1        !                              LABSOL2.OPT
 2        !
 3        GSMATCH=LEQUAL,1,0
 4        CLUSTER=A_TRANSFER,,,LABSOL2B
```

```
 1        $ COBOL LABSOL2A
 2        $ MACRO LABSOL2B
 3        $ COBOL LABSOL1
 4        $ LINK/SHARE LABSOL2A,LABSOL2/OPT
 5        $ LIBRARY/CREATE/SHARE SHRLIB LABSOL2A
 6        $ LINK LABSOL1,SHRLIB/LIBRARY
 7        $ ASSIGN DISK$COURSE:[USER]LABSOL2A LABSOL2A
 8        $ RUN LABSOL1
```

(Sheet 3 of 3)

3.

```
1       *                                                LABSOL3A.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL3A.
5       *
6       *   This is a blockdata program which is linked and
7       *   installed as a writeable shareable image.  It is
8       *   shared by LABSOL3B.COB and LABSOL3C.COB
9       *
10      DATA DIVISION.
11      WORKING-STORAGE SECTION.
12      01  DATA_ARRAY_1 IS EXTERNAL.
13         02       DIM_1            OCCURS 50 TIMES.
14         03       IARRAY           PIC 999 COMP.
15      01  DATA_ARRAY_2 IS EXTERNAL.
16         02       DIM_2            OCCURS 50 TIMES.
17         03       JARRAY           PIC 999 COMP.
```

```
1       *                                                LABSOL3B.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL3B.
5       *
6       *   This program writes to a shared data area in the
7       *   installed shareable image called LABSOL3A.  The
8       *   program must be linked with LABSOL3A in order to
9       *   execute.
10      *
11      DATA DIVISION.
12      WORKING-STORAGE SECTION.
13      01  DATA_ARRAY_1 IS EXTERNAL.
14         02       DIM_1                    OCCURS 50 TIMES.
15         03       IARRAY           PIC 999 COMP.
16      01  DATA_ARRAY_2 IS EXTERNAL.
17         02       DIM_2                    OCCURS 50 TIMES.
18         03       JARRAY           PIC 999 COMP.
19      01  INUMBER          PIC 999 COMP    VALUE 0.
20      01  JNUMBER          PIC 999 COMP    VALUE 0.
21      01  I                PIC 999.
22      01  J                PIC 999.
23      PROCEDURE DIVISION.
24      BEGIN.
25          PERFORM VARYING I FROM 1 BY 1 UNTIL I > 50
26                  COMPUTE INUMBER = INUMBER + 5
27                  MOVE INUMBER TO IARRAY(I)
28          END-PERFORM
29          PERFORM VARYING J FROM 1 BY 1 UNTIL J > 50
30                  COMPUTE JNUMBER = JNUMBER + 5
31                  MOVE JNUMBER TO JARRAY(J)
32          END-PERFORM
33          DISPLAY 'Write to global data area completed.'
34          STOP RUN.
```

(Sheet 1 of 2)

```
1       *                                                    LABSOL3C.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL3C.
5       *
6       *   This program reads the data stored in the installed
7       *   shareable image called LABSOL3C. It must be linked
8       *   with the shareable image to execute properly.
9       *
10      DATA DIVISION.
11      WORKING-STORAGE SECTION.
12
13      01  DATA_ARRAY_1 IS EXTERNAL.
14          02       DIM_1                    OCCURS 50 TIMES.
15          03       IARRAY           PIC 999 COMP.
16      01  DATA_ARRAY_2 IS EXTERNAL.
17          02       DIM_2                    OCCURS 50 TIMES.
18          03       JARRAY           PIC 999 COMP.
19      01  I                PIC 99 COMP.
20      01  J                PIC 99 COMP.
21      01  K                PIC 9(9) COMP.
22      01  DISP_I           PIC Z(8)9.
23      01  DISP_J           PIC Z(8)9.
24
25      PROCEDURE DIVISION.
26      BEGIN.
27      *
28      *   Compute and display mean of IARRAY
29          MOVE 0 TO K
30          PERFORM VARYING I FROM 1 BY 1 UNTIL I > 50
31             COMPUTE K = K + IARRAY(I)
32          END-PERFORM
33          COMPUTE K = K/50
34          MOVE K TO DISP_I
35          DISPLAY ' Mean of IARRAY = ' DISP_I
36      *
37      *   Compute and display mean of JARRAY
38          MOVE 0 TO K
39          PERFORM VARYING J FROM 1 BY 1 UNTIL J > 50
40             COMPUTE K = K + JARRAY(J)
41          END-PERFORM
42          COMPUTE K = K/50
43          MOVE K TO DISP_J
44          DISPLAY ' Mean of JARRAY = ' DISP_J
45          STOP RUN.
```

```
1       $ COBOL LABSOL3A
2       $ LINK/SHARE LABSOL3A
3       $ SET PROC/PRIV=SYSPRV
4       $ COPY LABSOL3A.EXE SYS$SHARE:LABSOL3A.EXE
5       $ PURGE SYS$SHARE:LABSOL3A.EXE
6       $ SET PROC/PRIV=NOSYSPRV
7       $ COBOL LABSOL3B, LABSOL3C
8       $ LINK LABSOL3B, SYS$INPUT/OPTIONS
9       SYS$SHARE:LABSOL3A/SHARE
10      $ LINK LABSOL3C, SYS$INPUT/OPTIONS
11      SYS$SHARE:LABSOL3A/SHARE
12      $ SET PROC/PRIV=CMKRNL
13      $ RUN SYS$SYSTEM:INSTALL
14      SYS$SHARE:LABSOL3A/WRITE/SHARE
15      /EXIT
16      $ RUN LABSOL3B
17      $ RUN LABSOL3C
18      $ RUN SYS$SYSTEM:INSTALL
19      SYS$SHARE:LABSOL3A/DELETE
20      /EXIT
21      SET PROC/PRIV=NOCMKRNL
```

(Sheet 2 of 2)

Measuring and Improving Performance

File Locations

On-line files are provided to help you master this module. The files are located in the directory:

```
DISK$COURSE:[COURSE.V4PROG.COB.PERF]
```

For your convenience, your system manager may have created the following logical name equivalence:

```
DISK$COURSE:[COURSE.V4PROG.COB.PERF] = V4PROG$COB$PERF
```

Three types of files are provided:

1. The Module Programming Examples, which are also listed in this language workbook.
2. Files that you modify to complete the Laboratory Exercises.
3. Solutions to the Laboratory Exercises.

If you wish to modify and/or compile, link, and run any of the programs, first copy the file into your directory.

Example 1

This example illustrates the use of Run-Time Library routines to measure the performance of a program. The routines are used to collect and display information on the resource usage of the program.

❶ The call to LIB$INIT_TIMER stores the current values of the program statistics to be measured. Since no storage block was specified, the values are kept in storage space maintained by the RTL routines.

❷ Initializing the array consumes system resources. Notice that many page faults are incurred because the array was not accessed in the most efficient order.

❸ The call to LIB$SHOW_TIMER obtains the accumulated times and counts since the call to LIB$INIT_TIMER. Since no code or action-routine has been specified, the statistics are output to the terminal in ASCII format.

```
 1       *                                                LIBTIMER.COB
 2       IDENTIFICATION DIVISION.
 3       *
 4       PROGRAM-ID. LIBTIMER.
 5       *
 6       *   This program illustrates the use of the RTL
 7       *   performance measurement routines.
 8       *
 9       DATA DIVISION.
10
11       WORKING-STORAGE SECTION.
12       01  ARRAY.
13           02  DIM1          OCCURS 100 TIMES.
14           03  DIM2          OCCURS 100 TIMES.
15           05  IARRAY        PIC S9(9) COMP.
16       01  I                 PIC S9(9) COMP.
17       01  J                 PIC S9(9) COMP.
18       01  RESULT            PIC S9(9) COMP.
19
20       PROCEDURE DIVISION.
21       BEGIN.
22       *
23  ❶        CALL 'LIB$INIT_TIMER' GIVING RESULT.
24           IF RESULT IS FAILURE CALL 'LIB$STOP' USING BY VALUE RESULT.
25       *
26       *   Initialize the table
27           PERFORM VARYING J FROM 1 BY 1 UNTIL I > 100
28                   PERFORM VARYING I FROM 1 BY 1 UNTIL I > 100
29  ❷                        MOVE 0 TO IARRAY(I,J)
30                   END-PERFORM
31           END-PERFORM
32       *
33  ❸        CALL 'LIB$SHOW_TIMER' GIVING RESULT.
34           DISPLAY  'Usage values after table initialization'.
35           IF RESULT IS FAILURE CALL 'LIB$STOP' USING BY VALUE RESULT.
36           STOP RUN.
```

```
$ COBOL LIBTIMER
$ LINK LIBTIMER
$ RUN LIBTIMER
 ELAPSED: 00:00:00.11  CPU: 0:00:00.06  BUFIO: 0  DIRIO: 0  FAULTS: 80
Usage values after table initialization
$
```

Example 1 Measuring Performance with RTL Routines

Example 2

This program illustrates the use of a termination mailbox. When the subprocess is deleted, the system writes a termination message to the mailbox associated with a subprocess through the $CREPRC system service.

❶ SUBPROC.EXE is the name of the image to be executed by the created subprocess.

❷ The item list for $GETDVI consists of an item descriptor, followed by a zero longword. The second word of this item descriptor defines the value of the DVI$__UNIT symbolic code. To determine the value of a symbolic code:

a. Create the file DVIDEF.MAR:

```
$DVIDEF     GLOBAL
.END
```

b. Issue the DCL commands:

```
$ MACRO/LIST DVIDEF
$ TYPE DVIDEF
```

The symbol table contains the **hexidecimal** values of all DVI$ symbolic codes. The decimal equivalents are used in the VALUE clause.

❸ TERMMBX__MSG is an array that receives the first 13 longwords of the termination mailbox message. (The rest of the information is not important to this program.) The complete format of the mailbox message is provided with the description of the $CREPRC system service in the *VAX/VMS System Services Reference Manual.*

❹ The unit number for the termination mailbox is returned in the item list from $GETDVIW.

❺ The unit number of the termination mailbox is passed in the MBXUNT argument of the call to $CREPRC.

❻ Once the subprocess is created, the program issues a $QIO read to the termination mailbox to read the termination message.

❼ The elapsed CPU time and accumulated page faults of the subprocess (extracted from the mailbox message) are displayed on the terminal. Notice that the completion status of the subprocess is 1, which stands for SS$__NORMAL.

```
1       *                                                TERMMBX.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. TERMMBX.
5       *
6       *   This program illustrates the use of a termination
7       *   mailbox.  When the created subprocess exits, the
8       *   creating process reads the message from the
9       *   termination mailbox.
10      *
11      DATA DIVISION.
12      WORKING-STORAGE SECTION.
13
14      01 MBXCHAN          PIC 9(9) COMP.
15      01 PROCESS_ID               PIC 9(9) COMP.
16      01 PID                      PIC X(8).
17  (1) 01 SUB_IMAGE               PIC X(7)        VALUE 'SUBPROC'.
18  (2) 01 ITEM_LIST.
19        02  ITEM_UNIT.
20         03       PIC S9(4)       COMP VALUE IS 4.
21         03       PIC S9(4)       COMP VALUE IS 12.
22         03   POINTER VALUE REFERENCE UNIT_NUM.
23         03   POINTER VALUE REFERENCE UNIT_LENGTH.
24        02  TERMINATOR-ENTRY      PIC S9(9) COMP  VALUE 0.
25      01 UNIT_NUM         PIC 9(9) COMP.
26      01 UNIT_LENGTH              PIC S9(4) COMP.
27  (3) 01  TERMMBX_MSG.
28          02              PIC X(4).
29          02 EXIT_STAT    PIC 9(9) COMP.
30          02              PIC X(36).
31          02 CPU_TIME     PIC 9(9) COMP.
32          02 PG_FAULTS    PIC 9(9) COMP.
33          02              PIC X(28).
34      01 DISP_EXIT_STAT   PIC Z(9).
35      01 DISP_CPU_TIME    PIC Z(9).
36      01 DISP_PG_FAULTS   PIC Z(9).
37      01 DONE_FLAG        PIC S9(9) COMP.
38      01 IO$_READVBLK     PIC S9(9) COMP  VALUE EXTERNAL IO$_READVBLK.
39      01 STAT             PIC S9(9) COMP.
40      01 IOSB.
41          02  IOSB1       PIC S9(9) COMP.
42          02  IOSB2       PIC S9(9) COMP.
43
44      PROCEDURE DIVISION.
45      BEGIN.
46      *
47      *   Allocate an event flag
48          CALL 'LIB$GET_EF' USING BY REFERENCE DONE_FLAG
49                            GIVING STAT.
50          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
51
52      *   Create mailbox and assign channel
53          CALL 'SYS$CREMBX' USING BY VALUE 0
54                                  BY REFERENCE MBXCHAN
55                                  BY VALUE 0 0 0 0 0
56                            GIVING STAT.
57          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
58      *
59      *   Get mbx unit number for $CREPRC call
60  (4)     CALL 'SYS$GETDVIW' USING BY VALUE DONE_FLAG MBXCHAN 0
61                                  BY REFERENCE ITEM_LIST IOSB
62                                  BY VALUE 0 0 0
63                            GIVING STAT.
64          IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
65          IF IOSB1 IS FAILURE CALL 'LIB$STOP' USING BY VALUE IOSB1.
```

Example 2 Using A Termination Mailbox to Record Statistics of a Subprocess (Sheet 1 of 2)

```
66          *
67          *   Create subprocess
68      5       CALL 'SYS$CREPRC' USING BY REFERENCE PROCESS_ID
69                                      BY DESCRIPTOR SUB_IMAGE
70                                      BY VALUE 0 0 0 0 0 0 0 0
71                                              UNIT_NUM 0
72                                GIVING STAT.
73              IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
74          *
75          *   Convert process id to hexidecimal
76              CALL 'OTS$CVT_L_TZ' USING BY REFERENCE PROCESS_ID
77                                        BY DESCRIPTOR PID
78                                        BY VALUE 8 4
79                                  GIVING STAT.
80              IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
81              DISPLAY 'PID of created subprocess is ' PID.
82          *
83          *   Issue $QIO read to termmbx (and wait)
84      6       CALL 'SYS$QIOW' USING BY VALUE DONE_FLAG MBXCHAN IO$_READVBLK
85                                    BY REFERENCE IOSB
86                                    BY VALUE 0 0
87                                    BY REFERENCE TERMMBX_MSG
88                                    BY VALUE 84 0 0 0 0
89                               GIVING STAT.
90              IF STAT IS FAILURE CALL 'LIB$STOP' USING BY VALUE STAT.
91              IF IOSB1 IS FAILURE CALL 'LIB$STOP' USING BY VALUE IOSB1.
92              MOVE EXIT_STAT TO DISP_EXIT_STAT.
93              MOVE CPU_TIME TO DISP_CPU_TIME.
94              MOVE PG_FAULTS TO DISP_PG_FAULTS.
95              DISPLAY 'Subprocess terminated with exit status = ' DISP_EXIT_STAT.
96              DISPLAY 'CPU time used: ' DISP_CPU_TIME.
97              DISPLAY 'Page faults incurred: ' DISP_PG_FAULTS.
98              STOP RUN.
```

```
 1          *                                                SUBPROC.COB
 2          IDENTIFICATION DIVISION.
 3          *
 4          PROGRAM-ID. SUBPROC.
 5          *
 6          *   Executed by the subprocess created by TERMMBX.COB.
 7          *   Doesn't do much
 8          *
 9          DATA DIVISION.
10          WORKING-STORAGE SECTION.
11
12          01  DUMMY           PIC S9(9) COMP.
13
14          PROCEDURE DIVISION.
15          BEGIN.
16          *
17              DISPLAY 'START OF SUBPROC'.
18              PERFORM 1000 TIMES
19                      MOVE 200 TO DUMMY
20              END-PERFORM.
21              DISPLAY 'END OF SUBPROC'.
22              STOP RUN.
```

```
$ COBOL TERMMBX, SUBPROC
$ LINK TERMMBX
$ LINK SUBPROC
$ RUN TERMMBX
PID of created subprocess is 000001C0
Subprocess terminated with exit status =         1
CPU time used:        59
Page faults incurred:       193
$
```

Example 2 Using A Termination Mailbox to Record Statistics of a Subprocess (Sheet 2 of 2)

Example 3

This program adjusts the size of the process working set.

❶ The $ADJWSL increases or decreases the number of pages in the process working set.

❷ The SHOW WORKING_SET command in DCL displays the current working set limit and the maximum quota.

❸ Notice that the program cannot decrease the working set limit beneath the minimum established by the operating system.

❹ Similarly, the process working set cannot be expanded beyond the authorized quota.

```
 1       *                                                ADJUST.COB
 2       IDENTIFICATION DIVISION.
 3       *
 4       PROGRAM-ID. ADJUST.
 5       *
 6       *   This program illustrates how a program can control
 7       *   its working set size using the $ADJWSL system service.
 8       *
 9       DATA DIVISION.
10       WORKING-STORAGE SECTION.
11
12       01  ADJUST_AMT      PIC S9(9) COMP  VALUE 0.
13       01  NEW_LIMIT       PIC S9(9) COMP  VALUE 0.
14       01  OUT_ADJUST_AMT  PIC -ZZZ9.
15       01  OUT_NEW_LIMIT   PIC -ZZZ9.
16       01  RESULT          PIC S9(9) COMP.
17
18       PROCEDURE DIVISION.
19       BEGIN.
20       *
21           PERFORM VARYING ADJUST_AMT FROM -50 BY 10 UNTIL ADJUST_AMT > 70
22       *
23       *           Modify working set limit
24    (1)            CALL 'SYS$ADJWSL' USING BY VALUE ADJUST_AMT
25                                          BY REFERENCE NEW_LIMIT
26                                          GIVING RESULT
27                   IF RESULT IS FAILURE
28                           CALL 'LIB$STOP' USING BY VALUE RESULT
29                   END-IF
30       *
31                   MOVE ADJUST_AMT TO OUT_ADJUST_AMT
32                   MOVE NEW_LIMIT TO OUT_NEW_LIMIT
33                   DISPLAY  ' Modify working set by '  OUT_ADJUST_AMT
34                            '   New working set size = '  OUT_NEW_LIMIT
35           END-PERFORM.
36           STOP RUN.

$ SET WORKING_SET/NOADJUST
(2) $ SHOW WORKING_SET
  Working Set      /Limit= 150   /Quota= 200    /Extent= 300
  Adjustment disabled   Authorized Quota= 200  Authorized Extent= 300

$ COBOL ADJUST
$ LINK ADJUST
$ RUN ADJUST
 Modify working set by -  50   New working set size =   100
 Modify working set by -  40   New working set size =    60
 Modify working set by -  30   New working set size =    55
(3) Modify working set by -  20   New working set size =    55
 Modify working set by -  10   New working set size =    55
 Modify working set by     0   New working set size =    55
 Modify working set by    10   New working set size =    65
 Modify working set by    20   New working set size =    85
 Modify working set by    30   New working set size =   115
 Modify working set by    40   New working set size =   155
 Modify working set by    50   New working set size =   205
 Modify working set by    60   New working set size =   265
(4) Modify working set by    70   New working set size =   300
$
```

Example 3 Modifying the Process Working Set Limit

Example 4

This example uses the $LKWSET system service to lock pages into a process working set. The locked pages are unlocked at image exit, or could be explictily unlocked using the $ULWSET system service.

❶ The $EXPREG system service is used to add two pages to the program region. The starting and ending addresses of the added pages are returned in the LOCK_RANGE array.

❷ The LOCK_RANGE array containing the addresses of the pages to be locked is passed to the $LKWSET system service.

```
 1       *                                                LOCKER.COB
 2       IDENTIFICATION DIVISION.
 3       *
 4       PROGRAM-ID. LOCKER.
 5       *
 6       *   This program illustrates how the program region
 7       *   can be expanded, and the new pages locked into the
 8       *   working set.
 9       *
10       DATA DIVISION.
11       WORKING-STORAGE SECTION.
12
13       01  LOCK_RANGE.
14          02       LOCK-RANGE-1    PIC S9(9) COMP.
15          02       LOCK-RANGE-2    PIC S9(9) COMP.
16       01  MORE_PAGES      PIC S9(9) COMP  VALUE 2.
17       01  RESULT          PIC S9(9) COMP.
18
19       PROCEDURE DIVISION.
20       BEGIN.
21       *
22       *   Add two pages to program region
23  ❶        CALL 'SYS$EXPREG' USING BY VALUE MORE-PAGES
24                                   BY REFERENCE LOCK-RANGE
25                                   BY VALUE 0 0
26                               GIVING RESULT.
27           IF RESULT IS FAILURE CALL 'LIB$STOP' USING BY VALUE RESULT.
28       *
29       *   Lock new pages in working set
30  ❷        CALL 'SYS$LKWSET' USING BY REFERENCE LOCK-RANGE
31                                   BY VALUE 0 0
32                               GIVING RESULT.
33           IF RESULT IS FAILURE CALL 'LIB$STOP' USING BY VALUE RESULT.
34       *
35           DISPLAY 'P0 expanded and locked into working set'.
36           STOP RUN.
```

```
$ COBOL LOCKER
$ LINK LOCKER
$ RUN LOCKER
P0 expanded and locked into working set
$
```

Example 4 Locking Pages in a Process Working Set

VAX COBOL Programming Considerations

The following is a checklist of items that may improve VAX COBOL program performance.

1. Do not mix data types in arithmetic operations.
2. Provide proper scaling for data items used in arithmetic expressions.
3. Use COMP values for INDEX items and for subscripts.
4. Use PERFORM n TIMES rather than PERFORM VARYING.
5. Use COMP-3 or COMP values whenever possible. When they are used, make them the same size they will be in arithmetic expressions. Avoid numeric DISPLAY items in arithmetic expressions.

   COMP values are faster than COMP-3 values; DISPLAY values are the slowest.
6. If numeric DISPLAY items must be used, do not make their pictures larger than is necessary; that is, do not use S9(9) if S9(4) will suffice.
7. Use longword binary subscripts and index names.
8. Avoid using the COMPUTE statement.
9. Use GO TO DEPENDING ON instead of an IF GO TO series.
10. Use SEARCH ALL instead of SEARCH. SEARCH ALL performs a binary search.

For more information on optimizing VAX COBOL programs, refer to the *VAX COBOL User's Guide*.

Lab Exercises

1. The program LAB1.COB listed below accesses the array DATA_ARRAY in column-major order. Modify LAB1.COB to create a similar program that accesses the array in row-major order. Check to see which program runs with the least number of page faults and the least CPU time usage by issuing a SHOW STATUS command, running a program, and then issuing a second SHOW STATUS command.

Provided File

```
*                                                LAB1.COB
IDENTIFICATION DIVISION.
*
PROGRAM-ID.        LAB1.
*
*  This program accesses a large array in column-major
*  order.
*
DATA DIVISION.
WORKING-STORAGE SECTION.

01 DATA_ARRAY.
   02      DIM_1          OCCURS 256 TIMES.
   03      DIM_2          OCCURS 256 TIMES.
     04 IARRAY            PIC S9(9) COMP.
01 I              PIC S9(9) COMP.
01 J              PIC.S9(9) COMP.

PROCEDURE DIVISION.
BEGIN.
*
*  Set all elements to 1
   PERFORM VARYING J FROM 1 BY 1 UNTIL J > 256
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 256
       MOVE 1 TO IARRAY(I,J)
     END-PERFORM
   END-PERFORM.
   STOP RUN.
```

2. Modify the programs from Exercise 1 so that they use Run-Time Library routines to determine the number of page faults and the elapsed CPU time from within the program.

3. Write a program to do the following:

   a. Use the $ADJWSL system service to set your working set limit to its minimum level.

   b. Set each element of a two dimension array (160,160) to 1.

   c. Output the number of page faults and elapsed CPU time it took to access the array.

Lab Solutions

1.

```
1       *                                                LABSOL1.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL1.
5       *
6       *   This program accesses a large array in row-major
7       *   order.
8       *
9       DATA DIVISION.
10      WORKING-STORAGE SECTION.
11
12      01  DATA_ARRAY.
13          02      DIM_1           OCCURS 256 TIMES.
14          03      DIM_2           OCCURS 256 TIMES.
15            04 IARRAY             PIC S9(9) COMP.
16      01  I               PIC S9(9) COMP.
17      01  J               PIC S9(9) COMP.
18
19      PROCEDURE DIVISION.
20      BEGIN.
21      *
22      *   Set all elements to 1
23          PERFORM VARYING I FROM 1 BY 1 UNTIL I > 256
24             PERFORM VARYING J FROM 1 BY 1 UNTIL J > 256
25                MOVE 1 TO IARRAY(I,J)
26             END-PERFORM
27          END-PERFORM.
28          STOP RUN.
```

2.

```
1       *                                                    LABSOL2A.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL2A.
5       *
6       *   This program accesses a large array in column-major
7       *   order.  It also outputs the amount of CPU time
8       *   and the number of page faults incurred by the
9       *   program.
10      *
11      DATA DIVISION.
12      WORKING-STORAGE SECTION.
13
14      01  DATA_ARRAY.
15          02      DIM_1           OCCURS 256 TIMES.
16          03      DIM_2           OCCURS 256 TIMES.
17            04 IARRAY             PIC S9(9) COMP.
18      01  I               PIC S9(9) COMP.
19      01  J               PIC S9(9) COMP.
20      01  TIMER_PTR       PIC S9(9) COMP  VALUE 0.
21      01  CPU_TIME        PIC S9(9) COMP  VALUE 2.
22      01  PAGE_FAULTS     PIC S9(9) COMP  VALUE 5.
23
24      PROCEDURE DIVISION.
25      BEGIN.
26      *
27      *   Initialize performance measurment block
28          CALL 'LIB$INIT_TIMER' USING TIMER_PTR.
29      *
30      *   Set all elements to 1
31          PERFORM VARYING J FROM 1 BY 1 UNTIL J > 256
32             PERFORM VARYING I FROM 1 BY 1 UNTIL I > 256
33                  MOVE 1 TO IARRAY(I,J)
34             END-PERFORM
35          END-PERFORM.
36      *
37      *   Get and display CPU time and page faults
38          CALL 'LIB$SHOW_TIMER' USING TIMER_PTR CPU_TIME.
39          CALL 'LIB$SHOW_TIMER' USING TIMER_PTR PAGE_FAULTS.
40      *
41      *   Free performance measurement block
42          CALL 'LIB$FREE_TIMER' USING TIMER_PTR.
43          STOP RUN.
```

(Sheet 1 of 2)

```
1       *                                                    LABSOL2B.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL2B.
5       *
6       *   This program accesses a large array in row-major
7       *   order.  It also outputs the amount of CPU time
8       *   and the number of page faults incurred by the
9       *   program.
10      *
11      DATA DIVISION.
12      WORKING-STORAGE SECTION.
13
14      01  DATA_ARRAY.
15          02      DIM_1           OCCURS 256 TIMES.
16          03      DIM_2           OCCURS 256 TIMES.
17            04 IARRAY             PIC S9(9) COMP.
18      01  I               PIC S9(9) COMP.
19      01  J               PIC S9(9) COMP.
20      01  TIMER_PTR       PIC S9(9) COMP  VALUE 0.
21      01  CPU_TIME        PIC S9(9) COMP  VALUE 2.
22      01  PAGE_FAULTS     PIC S9(9) COMP  VALUE 5.
23
24      PROCEDURE DIVISION.
25      BEGIN.
26      *
27      *   Initialize performance measurment block
28          CALL 'LIB$INIT_TIMER' USING TIMER_PTR.
29      *
30      *   Set all elements to 1
31          PERFORM VARYING I FROM 1 BY 1 UNTIL I > 256
32             PERFORM VARYING J FROM 1 BY 1 UNTIL J > 256
33                  MOVE 1 TO IARRAY(I,J)
34             END-PERFORM
35          END-PERFORM.
36      *
37      *   Get and display CPU time and page faults
38          CALL 'LIB$SHOW_TIMER' USING TIMER_PTR CPU_TIME.
39          CALL 'LIB$SHOW_TIMER' USING TIMER_PTR PAGE_FAULTS.
40      *
41      *   Free performance measurement block
42          CALL 'LIB$FREE_TIMER' USING TIMER_PTR.
43          STOP RUN.
```

(Sheet 2 of 2)

3.

```
1       *                                                    LABSOL3.COB
2       IDENTIFICATION DIVISION.
3       *
4       PROGRAM-ID. LABSOL3.
5       *
6       *   This program illustrates how a program can control
7       *   its working set size using the $ADJWSL system service.
8       *
9       DATA DIVISION.
10      WORKING-STORAGE SECTION.
11
12      01  NEW_LIMIT       PIC S9(9) COMP  VALUE 0.
13      01  OUT_NEW_LIMIT   PIC -ZZZ9.
14      01  RESULT          PIC S9(9) COMP.
15      01  IND_X           PIC S9(9) COMP VALUE 0.
16      01  TIMER_PTR       PIC S9(9) COMP.
17      01  CPU_TIME        PIC S9(9) COMP  VALUE 2.
18      01  PAGE_FAULTS     PIC S9(9) COMP  VALUE 5.
19      01  DATA_ARRAY.
20          02       DIM_1           OCCURS 256 TIMES.
21          03       DIM_2           OCCURS 256 TIMES.
22            04 IARRAY              PIC S9(9) COMP.
23      01  I               PIC S9(9) COMP.
24      01  J               PIC S9(9) COMP.
25
26      PROCEDURE DIVISION.
27      BEGIN.
28      *
29          PERFORM VARYING IND_X FROM 1 BY 1 UNTIL IND_X > 5
30      *
31      *   Modify working set limit
32          CALL 'SYS$ADJWSL' USING BY VALUE -50
33                                  BY REFERENCE NEW_LIMIT
34                                  GIVING RESULT
35          IF RESULT IS FAILURE
36                  CALL 'LIB$STOP' USING BY VALUE RESULT
37          END-IF
38      *
39          MOVE NEW_LIMIT TO OUT_NEW_LIMIT
40          DISPLAY 'New working set size = ' OUT_NEW_LIMIT
41      *
42          END-PERFORM.
43      *
44      *   Initialize timer
45          CALL 'LIB$INIT_TIMER' USING TIMER_PTR.
46      *
```

(Sheet 1 of 2)

```
47        *    Set IARRAY(I,J) = 1
48             PERFORM VARYING J FROM 1 BY 1 UNTIL J > 160
49                PERFORM VARYING I FROM 1 BY 1 UNTIL I > 160
50                     MOVE 1 TO IARRAY(I,J)
51                END-PERFORM
52             END-PERFORM.
53        *
54        *    Get and display CPU time and page faults
55             CALL 'LIB$SHOW_TIMER' USING TIMER_PTR CPU_TIME.
56             CALL 'LIB$SHOW_TIMER' USING TIMER_PTR PAGE_FAULTS.
57        *
58        *    Free performance measurement block
59             CALL 'LIB$FREE_TIMER' USING TIMER_PTR.
60        *
61        *
62             PERFORM VARYING IND_X FROM 1 BY 1 UNTIL IND_X > 5
63        *
64        *    Modify working set limit
65             CALL 'SYS$ADJWSL' USING BY VALUE 50
66                                     BY REFERENCE NEW_LIMIT
67                                     GIVING RESULT
68             IF RESULT IS FAILURE
69                     CALL 'LIB$STOP' USING BY VALUE RESULT
70             END-IF
71        *
72             MOVE NEW_LIMIT TO OUT_NEW_LIMIT
73             DISPLAY 'New working set size = ' OUT_NEW_LIMIT
74        *
75             END-PERFORM.
76        *
77        *    Initialize timer
78             MOVE 0 TO TIMER_PTR
79             CALL 'LIB$INIT_TIMER' USING TIMER_PTR.
80        *
81        *    Set IARRAY(I,J) = 1
82             PERFORM VARYING J FROM 1 BY 1 UNTIL J > 160
83                PERFORM VARYING I FROM 1 BY 1 UNTIL I > 160
84                     MOVE 1 TO IARRAY(I,J)
85                END-PERFORM
86             END-PERFORM.
87        *
88        *    Get and display CPU time and page faults
89             CALL 'LIB$SHOW_TIMER' USING TIMER_PTR CPU_TIME.
90             CALL 'LIB$SHOW_TIMER' USING TIMER_PTR PAGE_FAULTS.
91        *
92        *    Free performance measurement block
93             CALL 'LIB$FREE_TIMER' USING TIMER_PTR.
94        *
95             STOP RUN.
```

(Sheet 2 of 2)