Migration Specialties International, Inc.
217 West 2nd Street, Florence, CO   81226-1403
719-784-9196
E-mail: *Info@MigrationSpecialties.com*
www.MigrationSpecialties.com

Bruce Claremont, July 2005

Using OpenVMS to Meet a Sarbanes-Oxley Mandate

Part 2: The DCL

INTRODUCTION

Part 1 of this series covered our participation in a Sarbanes-Oxley compliance effort on OpenVMS systems. It discussed the project specification, planning, and implementation. Part 2, which concludes the series, covers in detail how we took advantage of OpenVMS and DCL to attain our goals. Part 2 provides examples of some of the DCL code we used to implement our compliance strategies. The article highlights how well the OpenVMS operating system coupled with good coding practices lends itself to Sarbanes-Oxley compliance.

SARBANES-OXLEY PROJECT IMPLEMENTATION DETAILS

The following sections provide detailed looks into some of the code we used to meet our Sarbanes-Oxley objectives. A reasonable comfort level with DCL coding is presumed for those that choose to peruse this article.

Termination Control and Error Reports

Sarbanes-Oxley auditing requires batch processes to capture and report process errors. From the Sarbanes-Oxley perspective, capturing an error kicks off an audit event covering the error and how it was handled.

Based on this requirement, our objective was to have procedures automatically report errors. We achieved this by controlling how all procedures terminate. This section discusses the procedure template we developed to achieve our goals. The template was to be applied to all new procedures. The initialization and termination sections in the template were applied to all existing active procedures. We also provided a set of calls so that a program or procedure creating a dynamic procedure could easily insert the standard initialization and termination code. This template is based on a standardized DCL template originally developed by Migration Specialties.

The template is very simple and does not limit procedure functionality. It enforces structured programming techniques via a single point of entry and exit. It also enforces implementation of return status values and a standard error handling routine.

Figure 1: DCL Procedure Template

```
$ IF F$MODE() .EQS. "BATCH" THEN SET VERIFY    !Leave this as first line in procedure!
$!------------------------------------------
$! proc.COM
$!------------------------------------------
$!     procedure purpose
$!
$!     Created:  id, mmm-yyyy
$!
$!     Modified:  id, mmm-yyyy
$!     -
$!
$!------------------------------------------------------------------------------
$!     STANDARD PROCEDURE INITIALIZATION SECTION (Don't modify this code)
$!------------------------------------------------------------------------------
$!
$ @SYS$UTILITIES:PROC_OPEN.COM 'F$ENVIRONMENT("PROCEDURE")'
$ _STATUS = 1
$ ON ERROR THEN GOTO ERROR_TRAP         !Standard abort trapping.
$ ON CONTROL_Y THEN GOTO ERROR_TRAP
$ IF _BATCH THEN SAY = "!"
$!
$!------------------------------------------------------------------------------
$!     PROCEDURE BODY
$!------------------------------------------------------------------------------
       .
       .
       .
$!------------------------------------------------------------------------------
$!     STANDARD PROCEDURE TERMINATION CODE (Modify with care)
$!------------------------------------------------------------------------------
$!
$ END_PROCEDURE:
$!
$!     *** Any procedure specific termination code goes here. ***
$!
$ @SYS$UTILITIES:PROC_CLOSE.COM   !Standard procedure shutdown call.
$ EXIT '_STATUS'
$!
$ ERROR_TRAP:
$!
$!     *** Procedure specific error handling code goes here. ***
$!
$ IF .NOT. _BATCH
$   THEN
$      SAY _BELL, "Procedure: ", F$ENVIRONMENT("PROCEDURE")
$      ASK "Procedure aborted by a <Ctrl^Y> or error.  Enter 0 to continue: " _QST
$      IF _QST .NES. "0" THEN GOTO ERROR_TRAP
$   ELSE
$      OPEN /WRITE ERR SYS$SCRATCH:ERR'_UID'.TMP
$      WRITE ERR "BATCH PROCESS ERROR NOTIFICATION"
$      WRITE ERR ""
$      WRITE ERR "Procedure: ", F$ENVIRONMENT("PROCEDURE")
$      WRITE ERR "Log File:  ", _LOGFILE
$      WRITE ERR "User:      ", P%USER
$      WRITE ERR "Time:      ", F$TIME()
$      CLOSE ERR
$      MAIL /SUBJECT=-
         " Batch Error Notification: ''F$PARSE(F$ENVIRONMENT("PROCEDURE"),,, "NAME")'"-
        SYS$SCRATCH:ERR'_UID'.TMP "@SYS$UTILITIES:COM_ERRORS.DIS"
$      DELETE SYS$SCRATCH:ERR'_UID'.TMP;*
$  ENDIF
$!
$ CANCEL_PROCEDURE:
$!
$!     Procedure specific abort handling code goes here.
$!
$!------------------------------------------------------------------------------
$!     STANDARD ABORT HANDLING CODE (Don't mess with this)
$!------------------------------------------------------------------------------
$!
$!     If this is the primary procedure (depth=0) and it is a batch procedure,
$!     have it exit with an error status (2) if it has terminated abnormally.
$!
$ IF _BATCH .AND. F$ENVIRONMENT("DEPTH") .LE. 0
$   THEN
$      _STATUS = 2
$   ELSE
$      _STATUS = 3
$  ENDIF
$ GOTO END_PROCEDURE
```

Detailed Template Walk-Through

This section provides a detailed walk-through of the DCL procedure template shown in *Figure 1*. The template components are explained, as is the reasoning behind the template's structure. Before beginning, our development rules need to be stated.

Standardized Procedure Development Rules

- No EXIT statements are permitted within the body of the procedure. Only one EXIT statement appears in any procedure and its location is pre-set in the termination code.
- All procedure termination runs through the END_PROCEDURE section.
- If the procedure is terminated abnormally for any reason, it must run through the ERROR_TRAP section.
- If the procedure is cancelled due to a decision process within the procedure that needs to be treated as an error, it must run through the CANCEL_PROCEDURE section.
- Do not use the following commands:
  - SET NOVERIFY or F$VERIFY(0)

    Batch jobs must create log files. Turning off verification prevents data from being written to the log files. Hence, the instruction is not used in production procedures.
  - ON *condition* THEN CONTINUE

    If a procedure generates an error, it must report the problem. Continuing from an error condition without an auditable event is not permitted.
- Under no circumstances should the SET NOON command be used to disable error trapping within a procedure. Its sole purpose is to reset error trapping protocols in special circumstances. Any time the SET NOON command is used, it should be immediately followed by these two lines of code:

  ```
  $ ON ERROR THEN GOTO ERROR_TRAP          !Standard abort trapping.
  $ ON CONTROL_Y THEN GOTO ERROR_TRAP
  ```

  These statements preserve standard error trapping functionality in the procedure.
- Use the ON *condition* commands with care. When special traps are created in procedures, ensure that they use the standard error routine to terminate the procedure. When the need for a special trap has passed, insert the following instruction:

  ```
  $ SET NOON
  $ ON ERROR THEN GOTO ERROR_TRAP          !Standard abort trapping.
  $ ON CONTROL_Y THEN GOTO ERROR_TRAP
  ```
- All procedure calls nested within a command procedure should be immediately followed by this command line:

  ```
  $ IF $STATUS .EQ. 3 THEN GOTO CANCEL_PROCEDURE
  ```

  Example:

  ```
  $ @PAYROLL:SINKORSWIM.COM
  $ IF $STATUS .EQ. 3 THEN GOTO CANCEL_PROCEDURE
  ```

This command line allows nested procedures that have generated an error to exit in a controlled fashion, preserving the error report capability.

Procedure Initialization

```
$ IF F$MODE() .EQS. "BATCH" THEN SET VERIFY    !Leave this as first line in procedure!
$!----------------------------------------
$! proc.COM
$!----------------------------------------
$!     procedure purpose
$!
$!     Created:  id, mmm-yyyy
$!
$!     Modified:  id, mmm-yyyy
$!     - modification
$!
$!----------------------------------------------------------------------------
$!     STANDARD PROCEDURE INITIALIZATION SECTION (Don't modify this code)
$!----------------------------------------------------------------------------
$!
$ @SYS$UTILITIES:PROC_OPEN.COM 'F$ENVIRONMENT("PROCEDURE")'
$ _STATUS = 1
$ ON ERROR THEN GOTO ERROR_TRAP         !Standard abort trapping.
$ ON CONTROL_Y THEN GOTO ERROR_TRAP
$ IF _BATCH THEN SAY = "!"
$!
$!----------------------------------------------------------------------------
$!     PROCEDURE BODY
$!----------------------------------------------------------------------------
$!
```

Figure 2: Procedure initialization section.

The F$MODE() lexical function on the first line ensures that the procedure will turn on verification if it is running in a batch process.

Modifications to the initialization section should be limited to comments. Highlights are:

proc	Procedure Name
procedure purpose	A brief description of the procedure
id	Programmer ID
mmm-yyyy	Month and Year
modification	A brief description of the modification

The **PROC_OPEN.COM** procedure creates standard application global symbols. PROC_OPEN contains logic so that it is only run once per session. Thus, global symbols are not needlessly recreated each time a new procedure executes. PROC_OPEN creates symbols like the following:

```
$ SAY :== WRITE SYS$OUTPUT
$ ASK :== READ SYS$COMMAND /END_OF_FILE=CANCEL_PROCEDURE /PROMPT="
$ _BATCH == 0
$ IF F$MODE() .EQS. "BATCH" THEN _BATCH == 1
$ _INTERACTIVE == 0
$ IF F$MODE() .EQS. "INTERACTIVE" THEN _INTERACTIVE == 1
$ _BELL[0,8] == %X7
```

The **_STATUS** symbol is reserved. It is used to permit exiting of nested procedures in a controlled fashion when an error is encountered.

Procedure Body

Within the bounds of the coding rules listed at the beginning of this section, everything between the *Procedure Body* and *Standard Procedure Termination Code* comment lines belongs to the programmer.

Procedure Termination Code

```
$!
$!------------------------------------------------------------------------
$!     STANDARD PROCEDURE TERMINATION CODE (Modify with care)
$!------------------------------------------------------------------------
$!
$ END_PROCEDURE:
$!
$!     *** Any procedure specific termination code goes here. ***
$!
$      @SYS$UTILITIES:PROC_CLOSE.COM    !Standard procedure shutdown call.
$      EXIT '_STATUS'
$!
```

Figure 3: Procedure termination section.

This section of the template is where all procedure termination takes place. This is the only place in the procedure where an EXIT statement is allowed. Any code in the body of the procedure that exits normally should do so by executing a GOTO END_PROCEDURE command.

Any special instructions that need to be executed prior to procedure termination should appear prior to the PROC_CLOSED.COM call. For example, deletion of temporary work files might take place here as part of the procedure clean-up process. The code in this section is executed every time the procedure terminates, regardless of whether the termination is normal or abnormal, so be careful with any code added to this section. Abnormal terminations are discussed in the *Error Trapping* section.

Any system or application wide standard procedure termination functions are incorporated into the **PROC_CLOSE.COM** procedure.

Error Trapping

```
$ ERROR_TRAP:
$!
$!     *** Procedure specific error handling code goes here. ***
$!
$ IF .NOT. _BATCH
$   THEN
$      SAY _BELL, "Procedure: ", F$ENVIRONMENT("PROCEDURE")
$      ASK "Procedure aborted by a <Ctrl^Y> or error.  Enter 0 to continue:" -
                _QST
$      IF _QST .NES. "0" THEN GOTO ERROR_TRAP
$   ELSE
$      OPEN /WRITE ERR SYS$SCRATCH:ERR'_UID'.TMP
$      WRITE ERR "BATCH PROCESS ERROR NOTIFICATION"
$      WRITE ERR ""
$      WRITE ERR "Procedure: ", F$ENVIRONMENT("PROCEDURE")
$      WRITE ERR "Log File:  ", _LOGFILE
$      WRITE ERR "User:      ", P%USER
$      WRITE ERR "Time:      ", F$TIME()
$      CLOSE ERR
$       MAIL /SUBJECT=-
          "Batch Error Notification: ''F$PARSE(F$ENVIRONMENT("PROCEDURE"),,, "NAME")'"-
          SYS$SCRATCH:ERR'_UID'.TMP "@SYS$UTILITIES:COM_ERRORS.DIS"
$       DELETE SYS$SCRATCH:ERR'_UID'.TMP;*
$   ENDIF
$!
```

Figure 4: Procedure error trapping and reporting section.

The ERROR_TRAP section exists to provide a standard mechanism for capturing and reporting errors. It is the heart of the template. Refined error trapping and control is permitted in the body of the procedure. However, if the procedure is to be exited with an error status, the last statement executed in the body of the procedure must be GOTO ERROR_TRAP.

Any special recovery code that is not implemented in the body of the procedure can be placed immediately following the ERROR_TRAP label. The recommended coding standard is to implement specialized error trapping within the procedure body and then initiate termination by executing the GOTO ERROR_TRAP command.

Using this template, error handling for batch and interactive jobs occurs as follows:

- Interactive jobs display an error message on the user’s screen and pause for user confirmation before terminating.
- Batch jobs issue an e-mail message to designated personnel. Designated personnel are listed in the SYS$UTILITIES:COM_ERRORS.DIS mailing list.

Code to achieve this standard is implemented following the IF statement. Programmer modifications to this section are limited to the area between the ERROR_TRAP label and the IF statement.

Nested Procedure Control

```
$ CANCEL_PROCEDURE:
$!
$!      Procedure specific abort handling code goes here.
$!
$!-----------------------------------------------------------------------------
$!      STANDARD ABORT HANDLING CODE (Don't mess with this)
$!-----------------------------------------------------------------------------
$!
$!      If this is the primary procedure (depth=0) and it is a batch procedure,
$!      have it exit with an error status (2) if it has terminated abnormally.
$!
$       IF _BATCH .AND. F$ENVIRONMENT("DEPTH") .LE. 0
$         THEN
$              _STATUS = 2
$         ELSE
$              _STATUS = 3
$        ENDIF
$       GOTO END_PROCEDURE
```

Figure 5: Nested procedure termination control section.

The CANCEL_PROCEDURE section deals with nested procedure control. This is why nested procedure calls need to be immediately followed by the command line:

```
$ IF $STATUS .EQ. 3 THEN GOTO CANCEL_PROCEDURE
```

This section of code takes care of ensuring that nested procedures exit in a controlled fashion. This is also why the _STATUS symbol is reserved. If an error occurs in a nested procedure, the _STATUS value is set to 3. As the stack of nested procedures unwinds, each procedure in the stack uses the lexical function F$ENVIRONMENT("DEPTH") to determine if it was the first procedure called. When the first procedure is reached, _STATUS is set to 2 to force an error condition via the system symbol $STATUS.

Log File Capture

One of the items we wanted to include in batch job error messages was the batch job log file string. This information was important to the auditors and provided a convenient way to quickly locate log files containing error information. Capturing the log file string from within the procedure that was generating the log file proved to be a bit of a challenge. We developed the following code to capture the log file name string accurately. We placed the code in the PROC_OPEN.COM procedure, positioned so it was run with every procedure call.

```
$!     The following commands will acquire the log file name from within
$!     a running batch procedure.
$!
$!     Acquire the procedure file specification associated with the current
$!     batch job.
$!
$      MASTER = F$GETQUI("DISPLAY_FILE", "FILE_SPECIFICATION",, "THIS_JOB")
$!
$!     Build a default log file name string from the batch procedure specification.
$!
$      DEFAULT_LOG = MASTER - F$PARSE(MASTER,,, "TYPE") - F$PARSE(MASTER,,, "VERSION")
$!
$!     Acquire the actual log file specification.
$!
$      LOG = F$GETQUI("DISPLAY_ENTRY", "LOG_SPECIFICATION",, "THIS_JOB")
$!
$!     The log file specification may not be complete.  If a SUBMIT /LOG command
$!     was used without an explicit log file entry, LOG will contain a null
$!     string.  If a SUBMIT /LOG=LOG$: type command was used, LOG will
$!     contain an incomplete file specification.  The following F$PARSE command
$!     uses the default log name string in DEFAULT_LOG to fill in an incomplete
$!     log specification.
$!
$      _LOGFILE == f$parse("''_LOG'", "''_DEFAULT_LOG'", ".log")
```

Figure 6: Log file capture code.

Automating Procedure Searches

To ensure our error trapping and reporting changes worked correctly in all existing procedures, we needed to make certain that none of the existing code could circumvent our changes. To this end, we needed to scan all of the procedures for code that might cause problems.

Rote work is something at which computers excel, so we built an analysis procedure that would scan for code that needed to be examined further. Specifically, we were looking for the following items:

Table 1:Potential DCL Problem Code

DCL Code	**Description**
ON CONTROL_Y ON ERROR ON SEVERE_ERROR	Breaks forced by ON *condition* statements needed to be checked to ensure they complied with our error trap strategy. Errors traps needed to always exit via the ERROR_TRAP routine.
SET NOVERIFY F$VERIFY(0)	Batch logs must contain data; hence, verification cannot be turned off. Procedures shutting off verification needed to be updated.
SUBMIT /NORETAIN /NOLOG	SUBMIT statements needed to be reviewed to ensure that they did not disable the /LOG qualifier or /RETAIN=ERROR qualifier.
END_PROCEDURE CANCEL_PROCEDURE ERROR_TRAP _STATUS	These reserved labels and symbols names were scanned to ensure we did not have any conflicts in existing code.
.LOG	Any procedure containing a reference to the .LOG file extension was examined to ensure it was not deleting or modifying log files.
SET NOON	SET NOON command usage was reviewed and modified to preserve the functionality of the ON ERROR and ON CONTROL_Y commands in the standardized error handling code.
@	Procedure calls were updated to include the follow on statement: $ IF $STATUS .EQ. 3 THEN GOTO CANCEL PROCEDURE They were also reviewed to ensure the procedure was listed as active and to determine if it was static or dynamic.

Figure 7 provides an example of the type of search procedure we developed to scan for the code we were interested in. The actual procedure we used is not available for publication. Suffice it to say, it was longer and a bit more sophisticated. However, this example, which searches for occurrences of ON ERROR and ON CONTROL_Y nicely illustrates the basic process.

Figure 7: Search Procedure Example

```
$! SEARCH.COM
$!    This procedure provides an example of how to use DCL
$!    to search files for specific text and report the findings
$!    in a comma-delimited format suitable for import into a
$!    spreadsheet program.
$!
$     OPEN /WRITE REPORT SEARCH.CSV          !Create report file.
$     WRITE REPORT " SEARCH RESULTS, ", F$TIME()
$     WRITE REPORT " File, On Error, On Cntl_Y, Message"
$!
```

```
$ MAIN_LOOP:
$      ON_ERR_CNT = 0                          !Event flag
$      ON_CTLY_CNT = 0                         !Event flag
$      FILE = F$SEARCH("*.COM;")        !Get file name
$      IF FILE .EQS. "" THEN GOTO END
$!
$      OPEN /ERROR=ERR_RPT /READ INPUT 'FILE'  !Open file
$      WRITE SYS$OUTPUT "Searching ", FILE
$!
$ READ_LOOP:
$      READ /END_OF_FILE=END_READ /ERROR=ERR_RPT INPUT RECORD
$      RECORD = F$EDIT(RECORD, "COMPRESS, UPCASE")    !Format line
$      TSTREC = F$EDIT(RECORD, "COLLAPSE")            !Test line
$      RECLEN = F$LENGTH(RECORD)               !Line length
$      IF F$EXTRACT(0, 2, TSTREC) .EQS. "$!" .OR. -   !Skip comments
         F$LENGTH(TSTREC) .LT. 2 THEN GOTO READ_LOOP !and blank lines.
$!
$!     Check for ON ERROR and ON CONTROL_Y commands.
$!     If found, count the event.
$!
$      IF F$LOCATE("ON ERROR", RECORD) .NE. RECLEN -
         THEN ON_ERR_CNT = ON_ERR_CNT + 1
$      IF F$LOCATE("ON CONTROL_Y", RECORD) .NE. RECLEN -
         THEN ON_CTLY_CNT = ON_CTLY_CNT + 1
$      GOTO READ_LOOP                                 !Get next line
$!
$ END_READ:                                           !Report results
$      IF ON_ERR_CNT .EQ. 0 THEN ON_ERR_CNT = ""      !Convert 0 to blank.
$      IF ON_CTLY_CNT .EQ. 0 THEN ON_CTLY_CNT = ""    !Convert 0 to blank.
$      WRITE REPORT FILE, ",", ON_ERR_CNT, ",", ON_CTLY_CNT, ",", ""
$      GOTO NEXT_FILE
$!
$ ERR_RPT:                                            !Report error
$      WRITE REPORT FILE, ", ?", ", ?", ", Error on Open or Read"
$!
$ NEXT_FILE:
$      CLOSE INPUT          !Close current file
$      GOTO MAIN_LOOP       !Get next file
$!
$ END:
$      CLOSE REPORT
$      EXIT
```

The end result of the search procedure is a comma-delimited file that can be imported into a spreadsheet program. The resulting spreadsheet provides key information at a glance and can be used to track changes and testing results. Our sample procedure produced the following sample results.

Table 2: Sample search results.

SEARCH RESULTS	28-JUL-2005 15:27:34.68		
File	On Error	On Cntl_Y	Message
$PRODUCTS:[SYS$UTILITIES]$CDROM_CHECK.COM;19	1	1	
$PRODUCTS:[SYS$UTILITIES]$LADCP_CHECK.COM;16	1	1	
$PRODUCTS:[SYS$UTILITIES]ACCOUNTING.COM;35	1	1	

Automated Batch Queue Checking

In addition to having batch jobs automatically report errors, we also built a process that automatically scanned all production batch queues and reported any jobs listed as retained on error. This provided redundancy in error reports and audit trails, something auditors like to see.

The procedure automatically carries out the following tasks:

- Identifies all batch queues on the system.
- Scans all entries in each batch queue.
- Records information on all jobs that are retained on error.
- Builds and sends an e-mail message concerning jobs retained on error to designated parties.
- Builds and maintains a comma-delimited file listing all jobs that have been retained on error.

The procedure is designed to be submitted on a periodic basis. In the case of this specific client, the procedure was submitted once a day from a master scheduler procedure.

The procedure, shown in its entirety in *Figure 8*, is well documented with comments. It is a good example of the power of OpenVMS DCL code and the F$GETQUI lexical function. It amply demonstrates Migration Specialties procedure standardization techniques along with our software development skills and code quality.

Figure 8: Automated batch queue check utility.

```
$ IF F$MODE() .EQS. "BATCH" THEN SET VERIFY    !Leave this as first line in procedure!
$!---------------------------------------
$! BATCHQ_CHECK.COM
$!---------------------------------------
$!     This procedure surveys all batch queues on the system, scanning for
$!     jobs that have been retained on error.  If such jobs are found, an
$!     e-mail message is issued with appropriate information to the personnel
$!     listed in the SYS$UTILITIES:COM_ERRORS.DIS mailing list.
$!
$!     The procedure also generates the BATCH_ERR_yyyy_mmm.CSV log file in the
$!     SYS$LOGS area.  A record of the batch jobs retained on error is recorded
$!     in this comma-delimited file.  A new version of the file is created at
$!     the beginning of each month.  The file is configured for easy importing
$!     into a spreadsheet program.
$!
$!     NOTE:  The procedure ignores queues with names beginning with TEST.
$!            This prevents the logging of jobs retained on error in the
$!            test system batch queues.
$!
$!     Created:  MSI, AUG-2004
$!
$!------------------------------------------------------------------------------
$!     STANDARD PROCEDURE INITIALIZATION SECTION (Don't modify this code)
$!------------------------------------------------------------------------------
$!
$ @SYS$UTILITIES:PROC_OPEN.COM 'F$ENVIRONMENT("PROCEDURE")'
$ _STATUS = 1
$ ON ERROR THEN GOTO ERROR_TRAP         !Standard abort trapping.
$ ON CONTROL_Y THEN GOTO ERROR_TRAP
$ IF _BATCH THEN SAY = "!"
$!
$!------------------------------------------------------------------------------
```

```
$!     PROCEDURE BODY
$!--------------------------------------------------------------------------------
$!     Check for privileges necessary to operate correctly.
$!
$ OLDPRIV = F$SETPRV("OPER")
$ IF .NOT.F$PRIVILEGE("OPER")
$   THEN
$      SAY "Error: Must have SYSPRV to run this procedure."
$      GOTO ERROR_TRAP
$  ENDIF
$!
$ CNT = 1
$ DEFAULT = ""
$ ERR = 0
$!
$!     Cancel any outstanding wildcard contexts for F$GETQUE service.
$!
$ TEMP = F$GETQUI("CANCEL_OPERATION")
$!
$ NAME_LOOP:
$!     This section locates all of the batch queues currently
$!     defined on the system and records their names in symbols.
$!
$ QNAME'CNT' = F$GETQUI("DISPLAY_QUEUE", "QUEUE_NAME", "*", "BATCH")
$ IF QNAME'CNT' .EQS. "" THEN GOTO GET_INFO
$ IF F$EXTRACT(0, 4, QNAME'CNT') .EQS. "TEST" THEN GOTO NAME_LOOP  !Skip test queues.
$ CNT = CNT + 1
$ GOTO NAME_LOOP
$!
$ GET_INFO:
$!     If the first queue name picked up is null, then no batch queues
$!     exist.  If this is the case, the procedure exits.
$!
$ IF QNAME1 .EQS. "" THEN GOTO END_PROCEDURE
$!
$!     Generate batch error log file name.  _YYYY & _MMM are global symbols
$!     created by PROC_OPEN.
$!
$ RPT = "SYS$LOGS:BATCH_ERR_''_YYYY'_''_MMM'.CSV"
$!
$!     Open batch error log file for writing.  The naming convention ensures
$!     a new file will be created if it is the beginning of the month.
$!
$!     NOTE: Leading blank appears in title lines on comma-delimited file to
$!     force lines to beginning of file when SORT /NODUPLICATIONS command is
$!     run against it.
$!
$ IF F$SEARCH(RPT) .EQS. ""
$   THEN
$      OPEN /SHARE=WRITE /WRITE REPORT 'RPT'
$      WRITE REPORT " BATCH PROCESS ERROR LOG: ''_MMM'-''_YYYY'"
$      WRITE REPORT " Queue, Date, Time, Entry, Procedure, Log File"
$   ELSE
$      OPEN /APPEND /SHARE=WRITE REPORT 'RPT'
$  ENDIF
$!
$!     Open e-mail message text file.
$!
$ OPEN /WRITE EMAIL SYS$SCRATCH:BATCH_ERR'_UID'.TMP
$!
$!     Scan the queues identified in the NAME_LOOP section.
$!
$ TEMP = F$GETQUI("CANCEL_OPERATION")
$ CNT = 1
$!
$ INFO_LOOP:
$ IF QNAME'CNT' .EQS. "" THEN GOTO FINISH
$!
$!     If queue name is a logical, ignore it.  Only physical queues are examined.
$!
$ IF F$GETQUI("TRANSLATE_QUEUE", "QUEUE_NAME", QNAME'CNT') .EQS. QNAME'CNT' -
        THEN GOSUB GET_QINFO
$ CNT = CNT + 1
$ GOTO INFO_LOOP
$!
$ FINISH:
$ CLOSE REPORT
$ WRITE EMAIL ""
$ WRITE EMAIL "Total of ''ERR' jobs retained on error"
$ CLOSE EMAIL
$ SORT /NODUPLICATES 'RPT' 'RPT'  !Remove duplicate records from log file.
$ PURGE 'RPT'
$ IF ERR .NE. 0
$   THEN
$      MAIL /SUBJECT="VMS Batch Jobs Currently Retained on Error: ''ERR'" -
               SYS$SCRATCH:BATCH_ERR'_UID'.TMP "@SYS$UTILITIES:COM_ERRORS.DIS"
$  ENDIF
$ DELETE SYS$SCRATCH:BATCH_ERR'_UID'.TMP;*
$!
$!---------------------------------------------------------------------------
$!     STANDARD PROCEDURE TERMINATION CODE (Modify with care)
$!---------------------------------------------------------------------------
$!
$ END_PROCEDURE:
$!
$!     *** Any procedure specific termination code goes here. ***
$!
$ OLDPRIV = F$SETPRV(OLDPRIV)     !Reset process privileges.
$ IF F$TRNLNM("EMAIL") .NES. "" THEN CLOSE EMAIL
$ IF F$TRNLNM("REPORT") .NES. "" THEN CLOSE REPORT
$ @SYS$UTILITIES:PROC_CLOSE.COM   !Standard procedure shutdown call.
$ EXIT 'STATUS'
$!
$ ERROR_TRAP:
$!
$!     *** Procedure specific error handling code goes here. ***
$!
$ IF .NOT. _BATCH
$   THEN
$      SAY _BELL, "Procedure: ", F$ENVIRONMENT("PROCEDURE")
$      ASK "Procedure aborted by a <Ctrl^Y> or error.  Enter 0 to continue: " _QST
$      IF _QST .NES. "0" THEN GOTO ERROR_TRAP
$   ELSE
$      OPEN /WRITE ERR SYS$SCRATCH:ERR'_UID'.TMP
$      WRITE ERR "BATCH PROCESS ERROR NOTIFICATION"
$      WRITE ERR ""
$      WRITE ERR "Procedure: ", F$ENVIRONMENT("PROCEDURE")
$      WRITE ERR "Log File:  ", _LOGFILE
$      WRITE ERR "User:      ", P%USER
$      WRITE ERR "Time:      ", F$TIME()
$      CLOSE ERR
$      MAIL /SUBJECT=-
  "VMS Batch Error Notification: ''F$PARSE(F$ENVIRONMENT("PROCEDURE"),,, "NAME")'" -
        SYS$SCRATCH:ERR'_UID'.TMP "@SYS$UTILITIES:COM_ERRORS.DIS"
$      DELETE SYS$SCRATCH:ERR'_UID'.TMP;*
$  ENDIF
```

```
$!
$ CANCEL_PROCEDURE:
$!
$!     Procedure specific abort handling code goes here.
$!
$!------------------------------------------------------------------------------
$!     STANDARD ABORT HANDLING CODE (Don't mess with this)
$!------------------------------------------------------------------------------
$!
$!     If this is the primary procedure (depth=0) and it is a batch procedure,
$!     have it exit with an error status (2) if it has terminated abnormally.
$!
$ IF _BATCH .AND. F$ENVIRONMENT("DEPTH") .LE. 0
$   THEN
$      _STATUS = 2
$   ELSE
$      _STATUS = 3
$  ENDIF
$ GOTO END_PROCEDURE
$!
$!****************************************************************************
$!                S U B R O U T I N E   S E C T I O N
$!****************************************************************************
$!
$ GET_QINFO:
$!     Get description and current status of queues.
$!     On Entry:
$!       CNT: ID for queue being checked.
$!
$ QDESC = ""  !Queue description
$ QSTATUS = ""      !Queue status: Online, Offline, Closed, Unknown
$ QERR = 1   !Queue error count
$!
$!     Get queue information and set context for obtaining job information.
$!
$ IF F$GETQUI("DISPLAY_QUEUE", "QUEUE_CLOSED", QNAME'CNT')
$   THEN
$      QSTATUS = "Closed"
$   ELSE
$      IF F$GETQUI("DISPLAY_QUEUE", "QUEUE_PAUSED", QNAME'CNT') -
              .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_PAUSING", QNAME'CNT') -
              .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_RESETTING", QNAME'CNT') -
              .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_STALLED", QNAME'CNT') -
              .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_STOPPED", QNAME'CNT') -
              .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_STOPPING", QNAME'CNT') -
              .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_UNAVAILABLE", QNAME'CNT')
$        THEN
$            QSTATUS = "Offline"
$        ELSE
$            QSTATUS = "Online"
$       ENDIF
$  ENDIF
$!
$!     Context for following F$GETQUI commands is set here!!!
$!
$ QDESC = F$GETQUI("DISPLAY_QUEUE", "QUEUE_DESCRIPTION", QNAME'CNT', "WILDCARD")
$!
$!     Look for any retained jobs on queue.
$!
$ JOB_LOOP:
$ ENTRY'QERR' = F$GETQUI("DISPLAY_JOB", "ENTRY_NUMBER",, "ALL_JOBS, RETAINED_JOBS")
$ IF ENTRY'QERR' .EQS. "" THEN GOTO END_JOBS
$!
```

```
$!     If job was retained on error, capture its information.
$!
$ IF .NOT. F$GETQUI("DISPLAY_JOB", "CONDITION_VECTOR",, "FREEZE_CONTEXT")
$   THEN
$      JOB'QERR'     = F$GETQUI("DISPLAY_JOB", "JOB_NAME",,"FREEZE_CONTEXT")
$      DATE_TIME     = F$GETQUI("DISPLAY_JOB", "JOB_COMPLETION_TIME",,
"FREEZE_CONTEXT")
$      DATE'QERR'    = F$CVTIME(DATE_TIME,, "DATE")
$      TIME'QERR'    = F$CVTIME(DATE_TIME,, "TIME")
$      PROC'QERR'    = F$GETQUI("DISPLAY_FILE", "FILE_SPECIFICATION",,
"FREEZE_CONTEXT")
$      DEFLOG'QERR'  = PROC'QERR' - F$PARSE(PROC'QERR',,, "TYPE")
$      LOG'QERR'     = F$GETQUI("DISPLAY_JOB", "LOG_SPECIFICATION",, "FREEZE_CONTEXT")
$      LOGFILE'QERR'= F$PARSE(LOG'QERR', DEFLOG'QERR', ".LOG")
$      QERR          = QERR + 1
$  ENDIF
$ GOTO JOB_LOOP
$!
$ END_JOBS:
$!
$!     If jobs retained on error were found, write results to
$!     e-mail message and report file.
$!
$ QERR = 1
$ IF ENTRY'QERR' .EQS. "" THEN GOTO END_WRITE
$!
$!     Output queue info.
$!
$ WRITE EMAIL F$FAO("!32AS !39AS !7AS", QNAME'CNT', QDESC, QSTATUS)
$!
$ WRITE_LOOP:
$!
$!     Output job info.
$!
$ WRITE EMAIL F$FAO("  !4SL !32AS !10AS !11AS", ENTRY'QERR', JOB'QERR', DATE'QERR',
TIME'QERR')
$ WRITE EMAIL F$FAO("       Procedure: !62AS", PROC'QERR')
$ WRITE EMAIL F$FAO("       Log File:  !62AS", LOGFILE'QERR')
$ WRITE EMAIL ""
$ WRITE REPORT QNAME'CNT', ",", DATE'QERR', ",", TIME'QERR', ",", ENTRY'QERR', -
    ",", PROC'QERR', ",", LOGFILE'QERR'
$ QERR = QERR + 1
$ IF ENTRY'QERR' .NES. "" THEN GOTO WRITE_LOOP
$!
$ END_WRITE:
$ ERR = ERR + QERR - 1
$ RETURN
$!
$!**********************************************************************
$!*                        DISCLAIMER                                  *
$!*    This software is provided as a service by Migration Specialties     *
$!*    International (MSI).  This software is provided as is and is       *
$!*    neither supported nor warrantied by MSI.  The customer may use     *
$!*    and modify this software at their own risk.                        *
$!*                 No warranties either expressed or implied.            *
$!**********************************************************************
```

About the author: *Mr. Bruce Claremont has a degree in Computer Science and has been working with OpenVMS since 1983. Bruce has extensive programming, project management, and system management experience. He has worked all sides of the fence, as a customer,* *software engineer, system manager, delivery specialist, project manager, and business owner. He founded Migration Specialties in 1992 and continues to deliver OpenVMS, software migration, and hardware emulation services. Bruce would like to help you achieve similar success in porting and supporting your applications. More information about Migration Specialties products and services can be found at www.MigrationSpecialties.com.*