OpenVMS Technical Journal V11

F$GETQUI to the Rescue

Bruce Claremont, Senior Consultant

Overview

The DCL lexical F$GETQUI is a powerful function that can be a bit confusing to deploy. This article demonstrates its usage via a useful DCL procedure designed to selectively delete entries from a queue. This article presumes a working knowledge of OpenVMS and DCL on the part of the reader.

F$GETQUI Lexical Function

Lexical functions provide easy access to a great deal of system information from the DCL command level. One of the most potent of these is the F$GETQUI function. It provides a way to interrogate the system for queue information, including batch entries, print entries, queue characteristics, and more.

Unlike most lexicals, which need only be called once to obtain information, the F$GETQUI function is iterative. Initial calls create a context that defines the information subsequent calls provide. Using the function may be a bit daunting at first, but mastering it provides a powerful tool for queue management and useful insights into the related system service calls that can be made from within programs.

Deleting Queue Entries

Anyone that has spent much time managing an OpenVMS system has run into the situation where they need to delete a large number of entries from an active queue. DCL provides the DELETE /ENTRY command to accomplish this. Unfortunately, DCL does not provide a means to specify a range of entries. Thus, to delete the following jobs,

```
Generic printer queue CPS_ANSI
Entry  Jobname         Username     Blocks  Status
  -----  -------         --------     ------  ------
     13  MIGRPG_LIST     CLAREMONT        42  Pending
     21  KIT             CLAREMONT        19  Pending
     23  CVTFILE         CLAREMONT        19  Pending
     27  CBL_RPT         CLAREMONT        19  Pending
```

one needs to enter the following command:

```
DELETE /ENTRY=(13,21,23,27)
```

This isn't too bad when there are only a few jobs, but when a print queue has been down for several hours or a batch job has failed repeatedly and had its entries retained using the /RETAIN=ERROR option, the list can get long. I've know managers that have deleted and recreated queues to clear them in these situations. Using F$GETQUI we can avoid such drastic measures, create a procedure that automates the process, and learn how F$GETQUI works.

CLEARQ

CLEARQ.COM is a procedure I created to automate clearing multiply entries from a queue. It allows the user to select a queue, then displays each entry on that queue in turn, and asks the user if they would like to delete the entry. It provides information on each entry to help the user determine if the job is a candidate for removal.

CLEARQ relies on F$GETQUI. The procedure is comprised of two main components, CLEARQ.COM and SELECTQ.COM. Both procedures appear at the end of this article. I will discuss CLEARQ in detail. SELECTQ also takes advantage of F$GETQUI to display a list of available queues to the user. It is similar in concept to CLEARQ. Determining the details of how it works is left as an exercise for the reader.

Procedure Initialization

```
$ ON ERROR THEN GOTO ERROR_TRAP           !Error trap.
$ ON CONTROL_Y THEN GOTO ERROR_TRAP       !User abort trap.
$ STATUS = 1                              !Default exit status value.
$ !Set terminal & process attributes.
$ @SYS$UTILITIES:SET_ATTRIBUTES.COM "''F$ENVIRONMENT("PROCEDURE")'"
$!
$ P1 = F$EDIT(P1, "COLLAPSE, UPCASE")     !Trim & upcase parameter.
$ IF F$EXTRACT(0, 3, P1) .EQS. "ALL" THEN P1 = "ALL_JOBS"
$ P2 = F$EDIT(P2, "COLLAPSE, UPCASE")     !Trim & upcase parameter.
$ ERRMSG = ""
$ FOUND = ""
$ _QUEUE = ""
```

Figure 1: Procedure initialization code

The block of code in *Figure 1* initializes the environment. It does the following:

- Creates and initializes variables.
- Edits optional user parameters P1 and P2.

The procedure SET_ATTRIBUTES.COM and my philosophy on code development are discussed in detail in the article *Simplification Thru Symbols*, available in OpenVMS Technical Journal V10.

Queue Selection

```
$! Obtain or validate queue name.  Validated queue name is returned in
$! global symbol _QUEUE.
$!
$ @SYS$UTILITIES:SELECTQ.COM 'P2'
$ IF $STATUS .EQ. 3 THEN GOTO CANCEL_PROCEDURE
$ IF _QUEUE .EQS. ""
$   THEN
$         SAY "Queue not found: ", _QUEUE
$         GOTO ERROR_TRAP
$  ENDIF
```

Figure 2: Queue selection call.

Having initialized our variables, we need to select the queue upon which we wish to act. The SELECTQ.COM procedure fulfills two roles in this regard. If the user provides a queue name via P2, SELECTQ verifies the entry is a valid, available queue. If P2 is invalid or the user does not provide a queue name, SELECTQ presents a list of available queues from which the user can choose.

SELECTQ takes full advantage of F$GETQUI functionality. The procedure appears in its entirety at the end of this article. The lessons imparted in our discussion of CLEARQ will help you understand how SELECTQ works.

Tip: SELECTQ passes back the queue name in the global symbol _QUEUE. If a local symbol called _QUEUE is also defined, it will take precedence and interfere with the operation of SELECTQ and CLEARQ (guess how I know this). This is one of those simple, subtle problems that can be difficult to identify. Adding code to check for and address conflicting local symbols would resolve the issue. Sounds like material for another article.

Setting Context

```
$!      Cancel any outstanding wildcard contexts for F$GETQUI service.
$!
$ TEMP = F$GETQUI("CANCEL_OPERATION")
```

Figure 3: Clearing context.

Setting context is key to using F$GETQUI. Before context is set, the CANCEL_OPERATIONS call ensures context is cleared from any prior F$GETQUI calls.

Setting context is not required for all F$GETQUI calls. For example, JOB_ENTRY calls do not require context to be set. However, for CLEARQ, we need context established to ensure subsequent F$GETQUI calls return information specific to the queue selected.

```
$ IF F$GETQUI("DISPLAY_QUEUE", "QUEUE_CLOSED", _QUEUE)
$   THEN
$     QSTATUS = "Closed"
$   ELSE
$     IF   F$GETQUI("DISPLAY_QUEUE", "QUEUE_PAUSED", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_PAUSING", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_RESETTING", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_STALLED", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_STOPPED", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_STOPPING", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_UNAVAILABLE", _QUEUE)
$        THEN
$          QSTATUS = "Offline"
$        ELSE
$          QSTATUS = "Online"
$        ENDIF
$  ENDIF
$!
$!      Context for following F$GETQUI commands is set here!!!
$!
$ QDESC = F$GETQUI("DISPLAY_QUEUE", "QUEUE_DESCRIPTION", _QUEUE, "WILDCARD")
```

Figure 4: Setting context.

The DISPLAY_QUEUE calls in the IF-THEN-ELSE block do not establish context. That means a subsequent DISPLAY_JOB or DISPLAY_FILE call may not display information from the desired queue or may issue an error. The purpose of these calls is to establish the status of the queue.

The DISPLAY_QUEUE/WILDCARD combination at the end of *Figure 4* establishes context. Subsequent F$GETQUI calls will return information within the context of the queue we have selected, which is identified by _QUEUE.

```
$ JOB_LOOP:
$!
$ ENTRY = F$GETQUI("DISPLAY_JOB", "ENTRY_NUMBER",, "''P1'")
$ IF ENTRY .EQS. "" THEN GOTO END_PROCEDURE
$ FOUND = 1
```

Figure 5: Listing entries.

Having established context, the procedure will use it to return information about each entry present on the selected queue. The DISPLAY_JOB/ENTRY_NUMBER call shown in *Figure 5* returns the entry number of the first job listed on the queue. Each pass through JOB_LOOP will pick up the next entry on the queue. Note there is no need to specify the queue name at this point.

The P1 parameter can be blank or specify the parameter ALL_JOBS. ALL_JOBS is only relevant to accounts with OPER privilege. By default, F$GETQUI will only return information on entries belonging to the UIC of the calling process. This is a handy feature. It allows unprivileged users to use CLEARQ without danger of them deleting entries that do not belong to them. If a privileged user runs CLEARQ without specifying ALL_JOBS, they too only see the entries that belong to them. Specifying ALL_JOBS allows a privileged user to see all entries on the queue.

```
$ JOB_STATUS = ""
$ IF F$GETQUI("DISPLAY_JOB", "JOB_ABORTING",, "FREEZE_CONTEXT") THEN -
      JOB_STATUS = JOB_STATUS + "Aborting, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_EXECUTING",, "FREEZE_CONTEXT") THEN -
      JOB_STATUS = JOB_STATUS + "Executing, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_HOLDING",, "FREEZE_CONTEXT") THEN -
      JOB_STATUS = JOB_STATUS + "Holding, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_INACCESSIBLE",, "FREEZE_CONTEXT") THEN -
      JOB_STATUS = JOB_STATUS + "Inaccessible, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_PENDING",, "FREEZE_CONTEXT") THEN -
      JOB_STATUS = JOB_STATUS + "Pending, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_REFUSED",, "FREEZE_CONTEXT") THEN -
      JOB_STATUS = JOB_STATUS + "Refused, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_RETAINED",, "FREEZE_CONTEXT") THEN -
      JOB_STATUS = JOB_STATUS + "Retained, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_STALLED",, "FREEZE_CONTEXT") THEN -
      JOB_STATUS = JOB_STATUS + "Stalled, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_STARTING",, "FREEZE_CONTEXT") THEN -
      JOB_STATUS = JOB_STATUS + "Starting, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_SUSPENDED",, "FREEZE_CONTEXT") THEN -
      JOB_STATUS = JOB_STATUS + "Suspended, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_TIMED_RELEASE",, "FREEZE_CONTEXT") THEN -
      JOB_STATUS = JOB_STATUS + "Timed Release, "
$!      !Get rid of trailing comma.
$ JOB_STATUS = F$EXTRACT(0, F$LENGTH(JOB_STATUS) - 2, JOB_STATUS)
$!
$ SAY _ESC, "[3;1f"
$ SAY "  Job:       ", _CEOL, _BOLD, -
   F$GETQUI("DISPLAY_JOB", "JOB_NAME",,"FREEZE_CONTEXT"), _CANCEL
$ SAY "  Entry:     ", _CEOL, _BOLD, ENTRY, _CANCEL
$ SAY "  Owner:     ", _CEOL, _BOLD, -
   F$GETQUI("DISPLAY_JOB", "USERNAME",, "FREEZE_CONTEXT"), _CANCEL
$ SAY "  Submitted: ", _CEOL, _BOLD, -
   F$GETQUI("DISPLAY_JOB", "SUBMISSION_TIME",, "FREEZE_CONTEXT"), _CANCEL
$ SAY "  File:      ", _CEOL, _BOLD, -
   F$GETQUI("DISPLAY_FILE", "FILE_SPECIFICATION",, "FREEZE_CONTEXT"), _CANCEL
$ SAY "  Status:    ", _CEOL, _BOLD, JOB_STATUS, _CANCEL
```

Figure 6: Obtaining entry (job) information.

Figure 6 demonstrates F$GETQUI DISPLAY_JOB and DISPLAY_FILE calls obtaining entry information that will be displayed to the user. The FREEZE_CONTEXT parameter is important. It prevents each DISPLAY call from jumping to the next entry on the queue. Note that the first DISPLAY call (*Figure 5*) does not freeze context. All subsequent DISPLAY calls must freeze context to keep F$GETQUI positioned on the same entry.

```
$ SELECTION = ""
$ RETRY_SELECTION:
$!
$ SAY _BOLD, _ESC, "[12;1f", ERRMSG, _CANCEL
$ SAY _ESC, "[11;1f", _CEOL              !Clear selection line.
$ SAY _ESC, "[11;1fRemove Entry (Y/N) [N]: ", _CANCEL, SELECTION, -
   ESC, "[11;29f", _BOLD, "<Ctrl^Z>", _CANCEL, " Quit"
$ ASK "''_ESC'[11;25f" SELECTION
$ SAY _ESC, "[12;1f", _CEOS       !Clear to end of screen
$ ERRMSG = ""                     !Clear error message.
$ SELECTION = F$EDIT(SELECTION, "COLLAPSE, UPCASE")
$ IF SELECTION  .EQS. "" THEN SELECTION = "N"
$ IF SELECTION  .NES. "N" .AND. SELECTION .NES. "Y"
$   THEN
$     ERRMSG = "Invalid selection (Y/N)"
$     GOTO RETRY_SELECTION
$  ENDIF
$ IF SELECTION  .EQS. "Y"
$   THEN
$     DELETE /ENTRY='ENTRY'
$!    ERRMSG = "Entry removed: ''ENTRY'"
$  ENDIF
$ GOTO JOB_LOOP
```

Figure 7: To delete or not to delete.

The code in *Figure 6* obtains and displays entry information to the user. The code in *Figure 7* asks the user if the entry should be removed. Once the user has provided a valid response, CLEARQ takes the appropriate action, then loops back to the DISPLAY_JOB call in *Figure 5* and starts the process over again. The lack of the FREEZE_CONTEXT parameter in the F$GETQUI call in *Figure 5* ensures the next entry in the queue is returned. This loop continues until no more entries are found or the user aborts the utility with a *<Ctrl^Z>*.

Conclusion

So there you have it, a simple example that demonstrates the harnessing of a very powerful lexical function, F$GETQUI. I have used this function in many interesting ways, such as:

- Obtaining and acting on queue and job information
- Controlling queues
- Logging job information for audits

Once you understand how to utilize F$GETQUI effectively, you too will find myriad applications for it.

Code
CLEAR1.COM

```
$! CLEARQ.COM
$!
$!      This procedure allows a user to clear entries from a queue.  The
$!      procedure presents queue entries one at a time and allows the user
$!      to remove the entry from the queue via a Y/N query.
$!
$!      If a queue name is passed to the procedure, it is validated.  If no
$!      queue name is passed, it is prompted for.
$!
$!      On Entry:
$!       P1 - ALL_JOBS   (Optional: Valid for privileged user only)
$!                       (          Can be abbreviated to ALL.    )
$!       P2 - Queue name (Optional)
$!
$ ON ERROR THEN GOTO ERROR_TRAP           !Error trap.
$ ON CONTROL_Y THEN GOTO ERROR_TRAP       !User abort trap.
$ STATUS = 1                              !Default exit status value.
$ !Set terminal & process attributes.
$ @SYS$UTILITIES:SET_ATTRIBUTES.COM "''F$ENVIRONMENT("PROCEDURE")'"
$!
$ P1 = F$EDIT(P1, "COLLAPSE, UPCASE")     !Trim & upcase parameter.
$ IF F$EXTRACT(0, 3, P1) .EQS. "ALL" THEN P1 = "ALL_JOBS"
$ P2 = F$EDIT(P2, "COLLAPSE, UPCASE")     !Trim & upcase parameter.
$ ERRMSG = ""
$ FOUND = ""
$ _QUEUE = ""
$!
$! Obtain or validate queue name.  Validated queue name is returned in
$! global symbol _QUEUE.
$!
$ @SYS$UTILITIES:SELECTQ.COM 'P2'
$ IF $STATUS .EQ. 3 THEN GOTO CANCEL_PROCEDURE
$ IF _QUEUE .EQS. ""
$   THEN
$         SAY "Queue not found: ", _QUEUE
$         GOTO ERROR_TRAP
$  ENDIF
$!
$!      Cancel any outstanding wildcard contexts for F$GETQUI service.
$!
$ TEMP = F$GETQUI("CANCEL_OPERATION")
$!
$!      Get queue information and set context for obtaining job information.
$!
$ IF F$GETQUI("DISPLAY_QUEUE", "QUEUE_CLOSED", _QUEUE)
$   THEN
$     QSTATUS = "Closed"
$   ELSE
$     IF   F$GETQUI("DISPLAY_QUEUE", "QUEUE_PAUSED", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_PAUSING", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_RESETTING", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_STALLED", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_STOPPED", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_STOPPING", _QUEUE) -
      .OR. F$GETQUI("DISPLAY_QUEUE", "QUEUE_UNAVAILABLE", _QUEUE)
$        THEN
$          QSTATUS = "Offline"
$        ELSE
$          QSTATUS = "Online"
$        ENDIF
$  ENDIF
$!
```

```
$!      Context for following F$GETQUI commands is set here!!!
$!
$ QDESC = F$GETQUI("DISPLAY_QUEUE", "QUEUE_DESCRIPTION", _QUEUE, "WILDCARD")
$!
$ SAY _BOLD, _REVERSE, _ESC, "[1;1fREMOVE ENTRIES UTILITY", _CANCEL, " -
    (Provided by the wonderful people at MSI)"
$ SAY "Queue: ", _UNDERLINE, _QUEUE, _CANCEL, " <", QDESC, "> ", QSTATUS
$!
$!      This section locates all of the entries on the specified queue
$!      and displays them one at a time.  The user is prompted to remove
$!      each entry.
$!
$ JOB_LOOP:
$!
$ ENTRY = F$GETQUI("DISPLAY_JOB", "ENTRY_NUMBER",, "'P1'")
$ IF ENTRY .EQS. "" THEN GOTO END_PROCEDURE
$ FOUND = 1
$!
$ JOB_STATUS = ""
$ IF F$GETQUI("DISPLAY_JOB", "JOB_ABORTING",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Aborting, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_EXECUTING",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Executing, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_HOLDING",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Holding, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_INACCESSIBLE",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Inaccessible, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_PENDING",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Pending, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_REFUSED",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Refused, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_RETAINED",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Retained, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_STALLED",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Stalled, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_STARTING",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Starting, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_SUSPENDED",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Suspended, "
$ IF F$GETQUI("DISPLAY_JOB", "JOB_TIMED_RELEASE",, "FREEZE_CONTEXT") THEN -
       JOB_STATUS = JOB_STATUS + "Timed Release, "
$!      !Get rid of trailing comma.
$ JOB_STATUS = F$EXTRACT(0, F$LENGTH(JOB_STATUS) - 2, JOB_STATUS)
$!
$ SAY _ESC, "[3;1f"
$ SAY "  Job:       ", _CEOL, _BOLD, -
   F$GETQUI("DISPLAY_JOB", "JOB_NAME",,"FREEZE_CONTEXT"), _CANCEL
$ SAY "  Entry:     ", _CEOL, _BOLD, ENTRY, _CANCEL
$ SAY "  Owner:     ", _CEOL, _BOLD, -
   F$GETQUI("DISPLAY_JOB", "USERNAME",, "FREEZE_CONTEXT"), _CANCEL
$ SAY "  Submitted: ", _CEOL, _BOLD, -
   F$GETQUI("DISPLAY_JOB", "SUBMISSION_TIME",, "FREEZE_CONTEXT"), _CANCEL
$ SAY "  File:      ", _CEOL, _BOLD, -
   F$GETQUI("DISPLAY_FILE", "FILE_SPECIFICATION",, "FREEZE_CONTEXT"), _CANCEL
$ SAY "  Status:    ", _CEOL, _BOLD, JOB_STATUS, _CANCEL
$!
$ SELECTION = ""
$ RETRY_SELECTION:
$!
$ SAY _BOLD, _ESC, "[12;1f", ERRMSG, _CANCEL
$ SAY _ESC, "[11;1f", _CEOL              !Clear selection line.
$ SAY _ESC, "[11;1fRemove Entry (Y/N) [N]: ", _CANCEL, SELECTION, -
   ESC, "[11;29f", _BOLD, "<Ctrl^Z>", _CANCEL, " Quit"
$ ASK "''_ESC'[11;25f" SELECTION
$ SAY _ESC, "[12;1f", _CEOS       !Clear to end of screen
$ ERRMSG = ""                     !Clear error message.
$ SELECTION = F$EDIT(SELECTION, "COLLAPSE, UPCASE")
$ IF SELECTION  .EQS. "" THEN SELECTION = "N"
```

```
$ IF SELECTION  .NES. "N" .AND. SELECTION .NES. "Y"
$   THEN
$     ERRMSG = "Invalid selection (Y/N)"
$     GOTO RETRY_SELECTION
$  ENDIF
$ IF SELECTION  .EQS. "Y"
$   THEN
$     DELETE /ENTRY='ENTRY'
$!    ERRMSG = "Entry removed: ''ENTRY'"
$  ENDIF
$ GOTO JOB_LOOP
$!
$ END_PROCEDURE:
$
$ SAY _ESC, "[1;1f", _CEOS
$ IF (.NOT. FOUND .AND. _QUEUE .NES. "") -
   THEN SAY "No entries found in queue ", _QUEUE
$ EXIT 'STATUS'
$!
$ ERROR_TRAP:
$
$ SAY _BELL
$ SAY "An error or <Ctrl^Y> has aborted this procedure."
$ CHECK_POINT:
$ IF F$MODE().NES."BATCH"
$   THEN
$         ASK "''_BELL'Enter 0 to exit the procedure: " CHK
$         IF CHK.NES."0" THEN GOTO CHECK_POINT
$  ENDIF
$!
$ CANCEL_PROCEDURE:
$
$ STATUS = 3
$ IF F$MODE().NES."BATCH" THEN SET TERMINAL/LINE_EDITING
$ GOTO END_PROCEDURE
```

SELECTQ.COM

```
$! SELECTQ.COM
$!-------------
$!      This procedure is used to allow users to select a queue.  The
$!      queue name is returned in the global symbol _QUEUE.
$!
$!      If a queue name is passed to the procedure, it is validated.
$!
$!      On Entry:
$!              P1 - Queue name (Optional)
$!
$       ON ERROR THEN GOTO ERROR_TRAP           !Error trap.
$       ON CONTROL_Y THEN GOTO ERROR_TRAP       !User abort trap.
$       STATUS = 1                              !Default exit status value.
$!      !Set terminal & process attributes.
$       @SYS$UTILITIES:SET_ATTRIBUTES.COM "''F$ENVIRONMENT("PROCEDURE")'"
$!
$       P1 = F$EDIT(P1, "COLLAPSE, UPCASE")
$       CNT = 1
$       DEFAULT = ""
$       ERRMSG = ""
$       IF F$TYPE(_QUEUE) .EQS. "" THEN _QUEUE == ""
$!
$!      Cancel any outstanding wildcard contexts for F$GETQUI service.
$!
$       TEMP = F$GETQUI("CANCEL_OPERATION")
$!
$!      If the user has not specified a queue name in P1, then display
$!      the menu heading information.
```

```
$!
$       IF P1 .EQS. ""
$         THEN
$              GOSUB HEADING
$              SAY _REVERSE, _BLINK, _ESC, -
                   "[12;4fGathering Queue Information - Please wait - ",-
                   F$TIME(), _CANCEL
$        ENDIF
$       NP1 = P1
$!
$ NAME_LOOP:
$!      This section locates all of the queues currently
$!      defined on the system and records their names in symbols.  If the user
$!      entered a queue name in P1, it is validated here.
$!
$       QNAME'CNT' = -
          F$GETQUI("DISPLAY_QUEUE", "QUEUE_NAME", "*")
$       IF QNAME'CNT' .EQS. "" THEN GOTO GET_INFO
$       IF P1 .NES. ""                          !Validate passed name?
$         THEN                                  !Yes
$              IF P1 .EQS. QNAME'CNT'               !Name match?
$                THEN                               !Yes
$                     GOSUB GET_QINFO
$                     IF QSTATUS'CNT' .NES. "Closed"  !Queue available?
$                       THEN                          !Yes
$                            SELECTION = CNT          !Select it.
$                            GOTO SETQ
$                       ELSE                          !No
$                            P1 = ""                  !Reset variables and
$                            CNT = 0                  !proceed as though no
$                            TEMP = F$GETQUI("CANCEL_OPERATION")
$                        ENDIF                        !entry was made.
$               ENDIF
$        ENDIF
$       CNT = CNT + 1
$       GOTO NAME_LOOP
$!
$ GET_INFO:
$!      If the procedure has reached this point with a value in P1, then an
$!      invalid queue name was entered by the user or the specified queue was
$!      closed.  The menu header and an appropriate error message are
$!      displayed.
$!
$       IF NP1 .NES. ""
$         THEN
$              GOSUB HEADING
$              IF P1 .NES. ""
$                THEN
$                     ERRMSG = "Queue specified <''NP1> does not exist"
$                ELSE
$                     ERRMSG = "Queue specified <''NP1'>is closed"
$               ENDIF
$        ENDIF
$!
$!      If the first queue name picked up is null, then no queues
$!      exist.  A message to that effect is displayed and the procedure
$!      exits.
$!
$       IF QNAME1 .EQS. ""
$         THEN
$              SAY ""
$              SAY _BELL, _BELL, _BELL, _BOLD, "No queues available on ", -
                   "this system at this time", _CANCEL
$              GOTO CHECK_POINT
$        ENDIF
$!
$!      Get the description and current status of the queues found in the
$!      previous section.
```

```
$!
$       TEMP = F$GETQUI("CANCEL_OPERATION")
$       CNT = 1
$!
$ INFO_LOOP:
$       IF QNAME'CNT' .EQS. "" THEN GOTO DISPLAY_QUEUES
$       GOSUB GET_QINFO
$       CNT = CNT + 1
$       GOTO INFO_LOOP
$!
$ DISPLAY_QUEUES:
$!      Display information collected on queus.
$!
$       CNT = 1
$       ROW = 4
$       SAY _ESC, "[", ROW, ";1f", _CEOS        !Clear remainder of screen.
$!
$ DISPLAY_LOOP:
$       IF QNAME'CNT' .EQS. "" THEN GOTO GET_SELECTION
$       DCNT = F$FAO("!3SL", CNT)
$       SAY _ESC, "[", ROW, ";1f", _BOLD, DCNT, ". ", _CANCEL, -
            F$EXTRACT(0, 15, QNAME'CNT'), _ESC, "[", ROW, ";22f", -
            F$EXTRACT(0, 51, QDESC'CNT'), _ESC, "[", ROW, ";74f", -
            QSTATUS'CNT'
$       CNT = CNT + 1
$       ROW = ROW + 1
$!
$!      If there are more than 13 queues to display, pause the display at
$!      the bottom of the screen and give the user the opportunity to
$!      Continue, Redisplay, make a Selection, or quit.
$!
$       IF ROW .LE. 22 THEN GOTO DISPLAY_LOOP
$       IF QNAME'CNT' .EQS. "" THEN GOTO GET_SELECTION
$!
$!      Prompt for queue name and then verify that it was correctly entered.
$!
$ GET_SELECTION:
$       SELECTION = ""
$!$ RETRY_SELECTION:
$       SAY _BOLD, _ESC, "[24;1f", ERRMSG, _CANCEL
$       SAY _ESC, "[23;1f", _CEOL               !Clear remainder of line.
$       SAY "''_BLINK'''_ESC'[23;1fSelection>''_CANCEL' ", SELECTION, -
            _ESC, "[23;20f", _BOLD, "<Enter>", _CANCEL, " More, ", -
            _BOLD, "<R>", _CANCEL, " Redisplay, ", -
            _BOLD, "<Ctrl^Z>", _CANCEL, " Quit"
$       ASK "''_ESC'[23;12f" SELECTION
$       SAY _ESC, "[24;1f", _CEOL       !Clear error line.
$       ERRMSG = ""                     !Clear error message.
$       SELECTION = F$EDIT(SELECTION, "COLLAPSE, UPCASE")
$       IF SELECTION  .EQS. "R"         !Redisplay queue list.
$         THEN
$              GOSUB HEADING
$              GOTO DISPLAY_QUEUES
$        ENDIF
$       IF SELECTION  .EQS. ""          !Display more queues if available,
$         THEN                          !otherwise start list over.
$           IF QNAME'CNT' .EQS. "" THEN GOTO DISPLAY_QUEUES
$           ROW = 4
$           SAY _ESC, "[", ROW, ";1f", _CEOS        !Clear remainder of screen.
$           GOTO DISPLAY_LOOP
$          ENDIF
$!
$!      Verify that a valid queue selection has been entered.  If not,
$!      display an error message and prompt for another selection.
$!
$       IF F$TYPE(QNAME'SELECTION') .NES. "STRING"
$         THEN
$              ERRMSG = "Invalid Entry"
```

```
$               GOTO RETRY_SELECTION
$        ENDIF
$       IF QSTATUS'SELECTION' .EQS. "Closed"
$         THEN
$               ERRMSG = "Queue is Closed"
$               GOTO RETRY_SELECTION
$        ENDIF
$!
$ SETQ:
$       _QUEUE == QNAME'SELECTION'
$!
$ END_PROCEDURE:
$       SAY _ESC, "[1;1f", _CEOS
$       EXIT 'STATUS'
$ ERROR_TRAP:
$       SAY _BELL
$       SAY "An error or <Ctrl^Y> has aborted this procedure."
$ CHECK_POINT:
$       IF F$MODE().NES."BATCH"
$         THEN
$               ASK "'_BELL'Enter 0 to exit the procedure: " CHK
$               IF CHK.NES."0" THEN GOTO CHECK_POINT
$        ENDIF
$ CANCEL_PROCEDURE:
$       STATUS = 3
$       IF F$MODE().NES."BATCH" THEN SET TERMINAL/LINE_EDITING
$       GOTO END_PROCEDURE
$!
$!*****************************************************************************
$!               S U B R O U T I N E   S E C T I O N
$!*****************************************************************************
$!
$ HEADING:
$!      Display selection menu heading and text.
$!
$       SAY _CLEAR, _BOLD, _ESC, "#3", _ESC, "[1;1f''_NODE' QUEUE SELECTION MENU"
$       SAY               _ESC, "#4", _ESC, "[2;1f''_NODE' QUEUE SELECTION MENU"
$       SAY _UNDERLINE, _ESC, "[3;34fNode: ", _NODE, _CANCEL
$       SAY _UNDERLINE, -
          _ESC, "[3;3f#", -
          _ESC, "[3;6fQueue Name     ", -
          _ESC, "[3;22fDescription                                        ", -
          _ESC, "[3;73f Status ", _CANCEL
$       RETURN
$!
$!*****************************************************************************
$!
$ GET_QINFO:
$!      Get description and current status of queues.
$!      On Return:
$!               QSTATUS'CNT' = "Online"
$!                              "Offline"
$!                              "Closed"
$!                              "Unknown"
$!
$       IF F$TRNLNM(QNAME'CNT') .NES. ""       !Logical name check.
$         THEN
$           QDESC'CNT' = "Queue name also a logical name, unable to get desc"
$           QSTATUS'CNT' = "Unknown"
$         ELSE
$           QDESC'CNT' = F$GETQUI("DISPLAY_QUEUE", "QUEUE_DESCRIPTION",
QNAME'CNT')
$           IF F$GETQUI("DISPLAY_QUEUE", "QUEUE_CLOSED", QNAME'CNT')
$             THEN
$               QSTATUS'CNT' = 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'CNT' = "Offline"
$                 ELSE
$                   QSTATUS'CNT' = "Online"
$                ENDIF
$             ENDIF
$          ENDIF
$        RETURN
```

About the author: Mr. Bruce Claremont has been working with OpenVMS since 1983. Mr. Claremont has extensive programming, project management, and system management experience. He also likes mechanical toys. He founded Migration Specialties in 1992 and continues to deliver OpenVMS and application migration services along with hardware emulation solutions. More information about Migration Specialties products and services can be found at *www.MigrationSpecialties.com*.

For more information

More on DCL development:

- *Simplification Thru Symbols*
  http://h71000.www7.hp.com/openvms/journal/v10/simplificationthrusymbols.html
- *Simplifying Maintenance with DCL*
  http://h71000.www7.hp.com/openvms/journal/v9/simplifying_maintenance_with_dcl.html

For the latest issue of the OpenVMS Technical Journal, go to:
http://www.hp.com/go/openvms/journal.

Thought for the day:

If bad code killed its creator, we would have better software applications.

hp invent