# RECOVER IT YOURSELF WITH USER LOGGING

by: Diane Weir
Los Alamos National Laboratory

## INTRODUCTION

IMAGE logging is a good product that has proved to be an effective and accurate way to save interactive transactions for recovery and audit purposes. There is one shortcoming with the product in that it only logs transactions within the IMAGE domain. Some applications require that KSAM and MPE files be updated in an on-line system. How can these files be recovered? One answer is to use a recoverable program structure that not only posts the interactive transactions, but recovers them as well. The user logging facility is used to store the successful transactions to either tape or disc. This paper will discuss the recoverable program structure and the user logging subsystem.

## LOGGING AND RECOVERING TRANSACTIONS

### THE RECOVERABLE PROGRAM STRUCTURE

This program structure was developed because there was a need to save all update transactions for an on-line payroll system that used IMAGE, KSAM, and MPE files. A record is written to the log file for every successful update on the file sets. The record logged looks just like the screen's image that caused the update. The on-line system can be run in interactive mode using V/3000 with users at terminals entering their transactions, or in recovery mode. In recovery mode the log file is read instead of the terminal. The edits are re done to insure that the data files were properly restored, then the transactions are posted to the backup copies of the files. If any of the edits fail the program aborts; the recovery is probably being run against the wrong set of files or the date/time parameters are wrong. The log file contains before and after images of the screen for audit reporting purposes; for recovery only the after images are reposted on the change transactions. The deletes contain the before image of the screen prior to the deletion, the adds, the after image.

### THE USER LOG RECORD FORMATS

Although the logging subsystem uses several record types, the records to concern yourself with are the user records, coded two and seven. The first nine words of the 128 word record are reserved for the logging subsystem. The first two words contain the record number, the next, the checksum. The fourth word is important in that it contains the record code. Words five through seven contain the date and time. The eighth word holds the log id with the ninth word used to hold the length of the user area. The user area follows for the next 119 words. Records are added to the log file via the WRITELOG intrinsic. The file is in 128 word ASCII format that, through the use of the length parameter, allows transactions of various lengths to be logged. If the write to the log file has a user area of 119 words or less, the transaction will physically be placed into one 2-record. If the 119 word limitation is exceeded, as many 7-records as needed to complete the operation are written. For example, if the length of the user area is 408 words, one will see one 2-record and three 7-records on the log for the request. This gives the user the flexibility needed for various uses of the log files.

When designing the application the 2-record was defined to contain certain control and audit information. In COBOL syntax the log records were defined as follows.

```
01  LOG-RECD.
    05  LOG-SYSTEM-AREA.
        10  FILLER              PIC X(6).
        10  LOG-REC-CD          PIC 9(4)     COMP.
        10  FILLER              PIC X(10).
    05  LOG-USER-AREA.
        10  LOG-USER            PIC X(8).
        10  LOG-SCREEN-CD       PIC XXX.
        10  LOG-BEF-AFT         PIC X.
        10  LOG-ACTION          PIC X.
        10  LOG-DATE            PIC S9(7)    COMP-3.
        10  LOG-TIME            PIC S9(5)    COMP-3.
        10  LOG-SCREEN-IMAGE    PIC X(218).
    05  LOG-CONT-AREA  REDEFINES  LOG-USER-AREA.
        10  LOG-CONT-SCREEN     PIC X(238).
```

The LOG-SYSTEM-AREA is the nine word area reserved by the user logging facility. The LOG-USER-AREA is the definition of the 2-record. It contains the user name obtained from the WHO intrinsic, the application's screen code, a before/after code, the screen's action code, the posting date and time, followed by the first 218 characters of data. If a continuation record was needed the LOG-CONT-SCREEN contains the 7-record's data. The before/after code is used primarily for audit reporting. An add transaction will contain only the after record; a delete, the before record. A change transaction will reflect both the before and after states. The screen image is remainder of the data written to the log file.

## THE MAIN PROGRAM

The on-line system contains a main menu program that prompts for passwords, opens the database, terminal, and formfile, and displays the menu. The sub programs actually update the files. The menu's function is to control the flow between the subprograms. The menu can be executed in interactive or recovery mode. The main program knows if recovery or interactive mode is desired via the use of run-time parameters. SW1 in COBOL was used if recovery mode was needed. Thus to run the program interactively, simply ;RUN PAO001, to run for recovery, :RUN PAO001;"PARM=%40000". The logic of the two modes is illustrated as follows.

### INTERACTIVE MODE

```
Turn echo off and ask for passwords
Open the database
Turn echo back on
Call the WHO intrinsic to find the
    user's name.
Gain access to the logging facility via
    OPENLOG
Open the formfile.
Open the terminal

READ-LOOP
Display the menu screen
Read the menu
Call the subprogram to service the
    request
    -or-
Go to the EXIT-ROUTINE if F8 was
    pressed.
Go to READ-LOOP
```

### RECOVERY MODE

```
Read the EDITOR file that has the from
    date/time and to date/time for the
    recovery process.
FOPEN the log file specifying an old,
    permanent file, opened for exclusive
    access.

READ-LOOP
Add 1 to the record number.
Call FREADDIR to read the file sequen-
    tially.
If the log record code is not a 2, skip
    the record.
Test the date/time in the log record to
    see that it fits the recovery para-
    meters.
If the log's time is less than the
    recovery time, read the next record.
If the log's time is greater than the
    recovery time, go to EXIT-ROUTINE.
Subtract 1 from the record number of
```

```
  the log file
Call the appropriate subprogram.
When the subprogram returns test the
  returning screen code for the end-
  -of-file flag.  If it is not set,
  subtract 1 from the record number and
  go to READ-LOOP.  If it is set go to
  EXIT-ROUTINE.
```

```
EXIT-ROUTINE
    Close the terminal and formfile
    Close the database and other files
    Terminate the access to the logging
      facility with CLOSELOG.
  Stop run.
```

```
EXIT-ROUTINE
    Close the log file via FCLOSE
    Close the database and other files
    Stop run.
```

When a system failure occurs in the middle of the day, the operator must first restore the files from the latest full and partial backup sets. Then the date and time of the last backup tape is entered into an EDITOR file along with the date and time of the system failure. This delimits the recovery process to the time parameters saved in the log file's 2-record. The operator then stops the logging process with the :LOG console command.

The database is opened for exclusive access while recovery is running. This insures that no processes are using the database until recovery is completed. The log file is also opened in exclusive mode as an extra safety measure. A stop of the logging system forces all buffers to be flushed to their media so the log file should be as complete as possible. If logging was not stopped prior to recovery the exlusive open of the log file will fail. The log files is then read using the FREADDIR call because the subprograms need to know where to begin processing on the log file. The record number is reduced by one prior to calling the subprogram so the subprogram can add one to the record number before reading the log file. This keeps the subprogram's read loop consistent.

## THE SUBPROGRAM

The subprogram's structure is described below. In interactive mode the screen is read and the data is edited. If the edits do not detect errors the database and other files are updated. For delete and add transactions, the screen image is added to the log file via WRITELOG directly from the screen image in working-storage. On change transactions, the "before" screen is re built and written to the log file before the updated screen image is logged. By comparing the "before" screen to the "after" screen on the audit report the changes can be isolated. The logic for the subprograms is outlined below.

### INTERACTIVE MODE

```
READ-LOOP
Read screen (VREADFIELDS,
             VFIELDEDITS,
             VGETBUFFER)

Edit the transaction
   (If errors perform VSETERROR
      then go to READ-LOOP)

Update the datasets and other files
```

### RECOVERY MODE

```
READ-LOOP
Add 1 to record number, then FREADDIR
     the log file.
Bypass any records whose code is not a 2
Test the "to date/time" against the rec-
     overy parameters.  If the time has
     expired or the end of file is found
     return to the menu.
See if the screen belongs to this
     subprogram, if not return to the
     main program.
See if the screen was too large to fit
     into one log record.  If so, cont-
     inue reading until the entire screen
     is reassembled.

Edit the transaction
     (If errors perform VSETERROR)

Update the datasets and other files
```

```
If add or delete, write screen to log
If change, build "before" screen
           write it to the log then
           log the "after" screen.

Initialize screen for next transaction.

Go to READ-LOOP                               Go to READ-LOOP
```

**VSETERROR ROUTINE**

If program is in interactive mode, call VSETERROR for the field, else abort.

In recovery mode, the routine to edit the screen's data and update the files is the same routine performed when recovery is run. There are not two separate programs to maintain when changes occur to the edit or update criteria. The same subprogram that updates the datasets, KSAM, and MPE files interactively also recovers those transactions. A word of caution. Since the edit routines are performed for the recovery to insure data integrety, any alterations to the edit criteria or the screen layout should be preceeded by s :STORE of the data- bases and other files updated by the system.

There is no need to delimit the logical transactions by special records written to the log file. IMAGE logging delimits transactions by DBBEGIN and DBEND calls. This is to prevent any incomplete updates from occurring. This is not needed in this type of structure. The screen is the logical transaction. One screen may update a variety of files but since the screen is being recovered instead of the records, special transaction delimiting records become unneces sary.

## THE COMMON AREA

The common area of the on-line system contains the V/3000 area, the database name, and the data needed for the logging and recovery. PP-USER comes from a call to WHO to determine the user's name. PP-LOG-INDEX is the log index returned from the call to OPENLOG. LOG-FILE-NUM is the file number for the log file when run for recovery; it is required to FREADDIR the log file. The PP-RECOVER-FLG indicates the mode, recovery or interactive, to the subprograms. PP-REC-NUM indicates where recovery is to begin on the log file. PP-SCREEN-CODE tells the main program the next screen to process or whether the subprogram reached the end of file or the time limit was exceeded. The recovery is terminated when log file ends or the log record's date and time exceed PP-RECOVER-TO-TIME.

```
01  COMMON-AREA.
    05  PP-D-BASE        PIC X(8).
    05  PP-USER          PIC X(8).
    05  PP-LOG-INDEX     PIC S9(9)  COMP.
    05  PP-REVCOVER-FLG  PIC S9(4)  COMP.
    05  PP-REC-NUM       PIC S9(9)  COMP.
    05  PP-SCREEN-CD     PIC X(4).
    05  PP-EOF-FLAG REDEFINES PP-SCREEN-CD
                         PIC X(4).
    05  PP-RECOVER-TO-TIME.
               10  PP-T-DATE     PIC S9(6).
               10  PP-T-TIME     PIC S9(4).
            05  LOG-FILE-NUM     PIC S9(4)  COMP.
            05  V-COM-AREA       PIC X(102).
```

## TESTING CONSIDERATIONS

How is testing conducted on a logging system? When testing occurs against a test database, the transactions should not be logged to the production log file. Instead the transactions are logged to a test log file. The log file identifier in main program is altered prior to the OPENLOG call. Also, the recovery should be tested if the screen layout was altered. This mandates that the FOPEN of the log file in recovery mode use the file name of the test log file, not the production log file. This can be accomplished through a file equation. Again the run parameter, SW2, was used to indicate whether testing was occurring. To test interactively one would :RUN PAO001;PARM=%20000, for testing recovery, :RUN PAO001;PARM=%60000.

## THE USER LOGGING SUBSYSTEM

### THE LOGGING PROCESS

The user logging subsystem allows for one shared file buffer per logging process regardless of the number of users accessing the log file. A write is performed on a log file via the WRITELOG intrinsic. One may log to either tape or disc. For disc logging, the log entries are loaded into the buffer area of the logging data segment. The records are written to disc when the buffer area becomes full or when certain intrinsics such as FLUSHLOG, BEGIN-LOG, or ENDLOG are called. Tape logging actually writes the log buffer to disc for a later transfer to tape. Transfers to tape occur simultaneously with writes to the disc log file because the two steps are controlled by separate processes. The reason for the two processes is so that the process that loads the buffer to disc can continue without interruption. The process that writes the transaction to tape can pause while a reel rewinds and another is mounted. This gives the logging process the capability to continue without interruption at the end of a tape volume.

IMAGE uses the user logging subsystem to record updates to the databses where logging is enabled. The user logging facility was written by a team in the MPE group to provide the framework for IMAGE logging. IMAGE records physical records updated by DBPUT, DBUPDATE, and DBDELETE calls if the logging on that database is active.

### GETTING STARTED WITH USER LOGGING

This section will deal with the commands and utilities used for the logging subsystem. All users accessing the user logging facility need to have logging, or LG, capability. The system manager needs to allocate LG capability to the account, then the account manager can allocate LG capability to himself and to the users accessing the interactive logging system.

```
:HELLO MANAGER.SYS
:ALTACCT PAYROLL;CAP=AM,AL,GL,OP,ND,SF,IA,BA,LG
:HELLO MGR.PAYROLL
:ALTUSER MGR;CAP=AM,AL,GL,OP,ND,SF,IA,BA,LG
:ALTUSER SALLY;CAP=ND,SF,IA,BA,LG
```

Estimate the size of the log files. They should be large enough to contain at least one day's worth of transactions. You may want to set the file size large enough to hold a week's worth of transactions if weekly audit reporting from the log file is desired. Build the log file with a record length of 128 words and a file code of LOG. Decide on a log identifier (log id). The log id is your link to the logging subsystem. Use the :GETLOG command to associate the log file with the log id, to tell the subsystem where logging is to occur, and to assign a password to the logging access. The password is not mandatory.

```
:BUILD PAD100;REC=128,5,F,ASCII;DISC=15000,16;CODE=LOG
:GETLOG PADLOG;LOG=PAD100,DISC;PASS=
```

### OPERATIONAL CONSIDERATIONS

The command to actually start the logging process is the :LOG console com mand. There is a problem with the console commands in that one must have been allowed the command in order to issue it from somewhere other than the console. The contributed utility ALLOWME will grant console command capability to users other than the owner of the console. The account manager was allowed the LOG command. OPERATOR.SYS was allowed both the LIMIT and LOG commands so that these can be controlled by the batch job running the SYSDUMPS.

```
:RUN ALLOWME.UTIL.SYS
Allowme Utility   V0.0  19 January 80
MGR.PAYROLL;COMMANDS=LOG
END OF PROGRAM
```

The jobstream for the SYSDUMPS sets the LIMIT to zero then stops the logging processes before a full or partial SYSDUMP. This allows the log file to be saved on the backup tape. After the SYSDUMP has completed, the jobstreams restart the logging processes and raise the limits back to normal. It is useful for the account manager to have access to the :LOG command and OP capability so that the logging process can be stopped and all files, including databases, can be stored prior to clearing the log file. OP capability allows a user to store a database without needing dangerous PM capability.

```
!JOB PARTIAL,OPERATOR/OPPASS.SYS/SYSPASS
!RUN ALLOWME.UTIL.SYS
!LIMIT 0,0
!CONTINUE
!LOG PADLOG,STOP
!FILE TP;DEV=7
!FILE LP;DEV=LP
!SYSDUMP *TP,*LP

10/20/83

Y
!LIMIT 2,16
!LOG PADLOG,RESTART
!EOJ
```

The system manager might need to alter the system configuration for the logging to work on your application. In the system table section of the SYSDUMP dialog, the manager defines the maximum number of logging processes allowed on the system at any one time and the maximum number of users per logging process. The system manager manual recommends 20 for both of these parameters but that might not be enough. When assigning these numbers remember that any IMAGE log ging performed on the system needs to be taken into account also.

One last operational consideration for the user logging facility is the OPERATOR.SYS startup procedure. Some shops stream a job and others use a UDC file. In the logon UDC for the console, the logging processes are restarted via the :LOG command.

```
STARTUP
OPTIONS LOGON,LIST
ALLOW OPERATOR.SYS;COMMANDS=CONSOLE
LOG PADLOG,RESTART
HEADOFF 6
STREAMS 10
JOBFENCE 4
OUTFENCE 4
STARTSPOOL LP
ALLOCATE EDITOR.PUB
 (etc) ...
```

## LOGGING COMMANDS

There are other logging commands that help one use the facility. :LISTLOG lists the active log identifiers on the system and their creators. :RELLOG del etes log identifiers from the user logging facility. :ALTLOG changes certain characteristics of the log id such as the log file name, the log destination, or or the logging password. :SHOWLOGSTATUS displays the status of all currently active log files. When the CIPER MIT was installed, the log identifiers were corrupted. The fix was to delete the bad log ids with :RELLOG and add the good ones back with :GETLOG. Fortunately, the log files were intact, just the identifiers were corrupted.

```
:LISTLOG        <<lists active log identifiers>>

    LOGID             CREATOR              LOGFILE
    PADLOG            MGR.PAYROLL          PAD100.PUB.PAYROLL

 :ALTLOG PADLOG;LOG=PAD100,TAPE     <<changes log characteristics>>
 :

 :SHOWLOGSTATUS            <<status display of active log processes>>

      LOGID          USERS     STATE          RECORDS
      PADLOG           0       INACT           225
```

```
:RELLOG PADLOG          <<deletes a log identifier>>
:
```

## THE LOG RECORDS AND THEIR FORMATS

There are nine record types in a log file. The format of the log records vary depending on the record type. Record type one is the openlog record. It is generated whenever a user accesses a logging system via the OPENLOG call. The three-record is the closelog record, generated when a user executes the CLOSELOG intrinsic. There is a start or restart record, code six. Records coded four and five are the transaction header and trailer record generated by the BEGINLOG and ENDLOG intrinsics. IMAGE uses these for DBBEGIN and DBEND calls. BEGINLOG and ENDLOG cause the logging buffer to be flushed to disc; so do DBBEGIN and DBEND calls. The nine record is a crash marker. When logging is restarted after a system failure while logging was active, recovery occurs on the log files. The crash marker tells the user logging subsystem where the crash occurred so it can recover itself. The user records, code two and seven, were discussed in the first section.

## INTRINSICS USED IN LOGGING

A write call to the logging subsystem uses a mode parameter. This parameter tells the logging system which action to take if the buffer becomes full prior to the write request. Mode one functions similar to no-wait I/O; the process continues after passing the request to the logging subsystem. Mode zero forces the process to wait until the logging system has processed the write to the log file. If the buffer is full and the write to the log file in mode zero, the buffer is written to disc and cleared. The entry is placed into the buffer, before control is returned to the calling program. The mode is also used for other calls to the logging system and operates in the same fashion.

Access to the logging facility is obtained via the OPENLOG intrinsic. The format is OPENLOG index, logid, password, mode, status. The index returned is used on subsequent calls to WRITELOG and CLOSELOG. The logid contains the log identifier, the password, the logging password. The mode indicates the wait request as discussed above. The status will contain error codes if the OPENLOG call failed.

The WRITELOG intrinsic format is WRITELOG index, data, length, mode, status. The index is the same index returned from the OPENLOG call. The data is the user data to be written to the user area of the log record. The length is the size of the data being passed. Again the mode is the wait/no-wait request.

The CLOSELOG intrinsic is used to stop access to the logging facility. Its Its format is CLOSELOG index, mode, status. Index, mode, and status are the same as for the WRITELOG intrinsic previously described.

## CONCLUSION

There are various methods available to users to recover lost interactive transactions. The method previously described is one way to approach the task. IMAGE logging is probably preferred since the programmer does not have to be involved with recovery. Unfortunately, IMAGE logging is not always the answer. There are files outside the IMAGE domain, KSAM and MPE files, that are updated via interactive programs that also need to be recovered. The user logging fac ility is an efficient answer to save those transactions that are critical to the application. The recoverable program structure described may be a useful tech nique since the chances of inconsistent results between two separate posting programs are eliminated. There is extra time required to develop and maintain the self-recovering programs, but the time is probably less than having one program post interactively and another post for recovery. There is a better chance of data consistency if one program does all the posting, be it inter active or recovery.

*BIOGRAPHICAL SKETCH of Diane Weir*

*I am currently employed by Los Alamos National Laboratory as a staff member using DEC equipment. Formerly, I was the data processing manager for a heavy construction firm using the HP 3000 for three years. It was in this capacity that this recovery technique and the User Logging subsystem was developed. I began in data processing in 1970 and have worked on IBM, Burroughs, Hewlett-Packard, and Digital hardware. I have a Bachelors of Business Administration from the University of Texas at El Paso.*

-----