North American Response Centers

HP 3000 APPLICATION NOTE

KSAM TOPICS:
Using Cobol II's Indexed I/O Module
and
Data Integrity & KSAM Files

HEWLETT PACKARD

June 15, 1986
Document P/N 5958-5824/2625

Two common KSAM topics, that result from questions received by the Response Centers, involve accessing KSAM files from COBOL II programs and dealing with data integrity problems in KSAM files. This Application Note will discuss both issues.

Using COBOL II's Indexed I/O Module

There are three ways to access KSAM files from COBOL. You can call the file system intrinsics such as FOPEN and FREADBYKEY directly; you can use the "CK" intrinsics, such as CKOPEN and CKREADBYKEY; or you can use COBOL II's Indexed I/O Module as described in this section.

When accessing KSAM files using the Indexed I/O Module, you use COBOL II statements instead of intrinsics, and the COBOL II compiler translates the statements into intrinsic calls for you.

The use of the Indexed I/O Module is discussed in the COBOL II manual. Note however that KSAM files are referred to there as COBOL Indexed Files (not as KSAM files).

Defining Indexed Files

In COBOL II, all files must be defined in the FILE-CONTROL paragraph within the INPUT-OUTPUT SECTION of the ENVIRONMENT DIVISION. The format for defining indexed files is as follows:

```
ENVIRONMENT DIVISION.
.
.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT file-name
   ASSIGN to "file-info-1" [,"file-info-2"] ...

   ;ORGANIZATION IS INDEXED

   [                        { SEQUENTIAL } ]
   [;ACCESS MODE IS         {   RANDOM   } ]
   [                        {   DYNAMIC  } ]

   ;RECORD KEY IS data-name-1 [ WITH DUPLICATES ]

   [;ALTERNATE RECORD KEY IS data-name-2 [ WITH DUPLICATES ] ] ...

   [;FILE STATUS IS stat-item].
```

All files defined in COBOL II must also have an FD file description within the FILE SECTION of the DATA DIVISION which contains the record description entry for the file. The record description entry defines the record to be associated with the file. Something to note here is that COBOL II does not have a data type which corresponds to KSAM key fields defined as REAL or LONG.

To give you an idea of what all of this looks like, here is an example of the ENVIRONMENT and DATA DIVISION constructs which define a KSAM file. (Included in the DATA DIVISION are the FILE STATUS data items which will be discussed later.)

```
.
.
ENVIRONMENT DIVISION.
.
.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT EMPLOYEE-FILE ASSIGN TO "EMPFILE.PUB.EMPACCT";
   ORGANIZATION IS INDEXED;
   ACCESS MODE IS DYNAMIC;
   RECORD KEY IS EMPLOYEE-NUM;
   ALTERNATE RECORD KEY IS LAST-NAME WITH DUPLICATES;
   FILE STATUS IS KSAM-STATUS.
.
.
DATA DIVISION.
FILE SECTION.
FD  EMPLOYEE-FILE.
01  EMPLOYEE-RECORD.                  )
    05  FILLER          PIC XX.       |
    05  EMPLOYEE-NUM    PIC X(9).     }  record description entry
    05  FILLER          PIC X.        |
    05  LAST-NAME       PIC X(16).    )
.
.
WORKING-STORAGE SECTION.
01  KSAM-STATUS.
    05  KSTAT-1         PIC X.
    05  KSTAT-2         PIC X.
.
.
```

Now let's take a look at each of the clauses of the FILE-CONTROL paragraph.

The SELECT clause is used to identify the KSAM file to be accessed and to assign it the file name by which it will be accessed in the program.

The ORGANIZATION clause is used to specify that the file is an INDEXED file, that is, a KSAM file.

The ACCESS MODE clause is used to specify how the file is to be accessed. SEQUENTIAL access is assumed if the ACCESS MODE clause is omitted. The three access modes will be described in the next section.

The RECORD KEY clause is used to name the data item within the file's record description entry which corresponds to the primary key. The ALTERNATE RECORD KEY clause is used to name the record descroption entry data item which corresponds to an alternate key; there is one ALTERNATE RECORD KEY clause per alternate key.

The FILE STATUS clause is used to name the data item, defined in the DATA DIVISION, to which status information regarding access to the files is to be returned. Whenever an operation is performed on the file, information as to the success or failure of the operation is put in this data area. The meaning of this status information is described in the FILE STATUS clause documentation in Section VII of the COBOL II manual.

Accessing and Modifying Indexed Files

The COBOL II statements which can be used to open and close indexed files are as follows:

```
     {INPUT  file-name-1 [,file-name-2] ...}
OPEN {OUTPUT file-name-3 [,file-name-4] ...}
     {I-O    file-name-5 [,file-name-6] ...}


CLOSE file-1 [WITH LOCK] [,file-2 [WITH LOCK] ] ...
```

All COBOL II allows to be specified when an indexed file is opened is the type of access for which the file is to be opened. INPUT indicates read only access; OUTPUT indicates write only access with all existing records deleted; and I-O access indicates read, write, and update access.

Because all that can be specified in an OPEN statement is the type of access to be allowed, COBOL II's Indexed I/O Module cannot be used to create KSAM files. Also, file equations must be used in order to override any defaults for where the file resides (,TEMP or ,SAVE), who may access the file (;EXC or ;EAR or ;SEMI or ;SHR), and whether or not dynamic locking is allowed (;LOCK or ;NOLOCK). Also note that file equations can be used to override the type of access specified in the OPEN (;IN or ;OUT or ;OUTKEEP or ;APPEND or ;INOUT or ;UPDATE).

File equations can also be used in conjunction with the CLOSE statement because the CLOSE statement closes files with the default disposition, unless a file equation specifies a different disposition (;SAVE or ;DEL).

The COBOL II statements which can be used to access indexed files are as follows:

```
                   [    {IS EQUAL TO     }          ]
                   [    {IS =            }          ]
START file-name    [KEY {IS GREATER THAN } data-name]
                   [    {IS >            }          ]
                   [    {IS NOT LESS THAN}          ]
                   [    {IS NOT <        }          ]

    [;INVALID KEY imperative-statement]

READ file-name RECORD [INTO identifier]

    [;AT END imperative-statement]
```

```
READ file-name [NEXT] RECORD [INTO identifier]

   [;AT END imperative-statement]


READ file-name RECORD [INTO identifier] [KEY IS data-name]

   [;INVALID KEY imperative-statement]
```

The way in which these statements can be used to access and modify KSAM files depends on the type of access -- SEQUENTIAL, RANDOM, or DYNAMIC -- for which the file was opened (as specified in the ACCESS MODE clause of the FILE CONTROL paragraph). Note that on HP machines there are really only two kinds of access -- sequential and dynamic -- because RANDOM is treated the same as DYNAMIC.

The START statement works the same in sequential and dynamic mode, and is like the FFINDBYKEY intrinsic. It positions the current record pointer within a specified key sequence, using full or generic keys and exact or approximate matching. It does not read a record, but is does establish a current key sequence. The KEY IS phrase allows you to specify a RECORD KEY or ALTERNATE RECORD KEY data item or any data item subordinate to them (provided that it starts at the beginning of a key) to be used for matching. If the KEY IS phrase is not used, the RECORD KEY data item is used.

The READ statement, as you can see, comes in three basic forms -- READ, READ...NEXT, and READ...KEY IS. The first form, READ, is used only in sequential mode and is like the FREAD intrinsic. If READ immediately follows OPEN, it will read the first record in primary key sequence. If READ immediately follows START, it will read the record to which START positioned the current record pointer. If READ follows another READ, it will read the next record in the current key sequence.

The second two forms of the READ statement, READ...NEXT and READ...KEY IS, are used only in dynamic mode. READ...NEXT in dynamic mode works just like READ in sequential mode, that is, like the FREAD intrinsic. READ...KEY IS, on the other hand, is equivalent to the FREADBYKEY intrinsic. READ...KEY IS reads the first record in a particular key sequence which has a particular key value. The KEY IS phrase allows you to specify which one of the RECORD KEY or ALTERNATE KEY data items is to be used for matching. If you omit the KEY IS phrase, the RECORD KEY data item is used.

From this description of indexed file access capabilities you can see that the Indexed I/O Module has some limitations when compared to the file system intrinsics. The main difference between the Indexed I/O Module and the intrinsics is that the Indexed I/O Module does not allow chronological access. That is, there are no equivalents to the FREADDIR, FPOINT, and FREADC intrinsics. The other difference is that the Indexed I/O Module does not allow access by logical record number because there is no equivalent to the FFINDN intrinsic.

The COBOL II statements which can be used to modify KSAM files are as follows:

```
WRITE record-name [FROM identifier-1]

  [;INVALID KEY imperative-statement]


DELETE file-name RECORD

  [;INVALID KEY imperative-statement]
```

```
REWRITE record-name [FROM identifier]

    [;INVALID KEY imperative-statement]
```

The WRITE, DELETE, and REWRITE statements can all be used in sequential or dynamic mode, but they work differently in the two modes.

The WRITE statement is like the FWRITE intrinsic, and is used to write a new record. When the access mode is sequential, records must be added in order of ascending primary key values, but when the access mode is dynamic, records may be added in any order.

The DELETE statement is used to delete a record. In sequential mode, a call to DELETE must be preceded by a call to READ, and DELETE will delete the record read by the READ statement. In dynamic mode, DELETE does not need to be preceded by READ. You specify the record you want deleted by putting its primary key value in the RECORD KEY data item of the file's record description entry. DELETE will delete the first record (in primary key sequence) which has that primary key value. DELETE does not affect the position of the current record pointer.

The REWRITE statement is used to update a record. In sequential mode, a call to REWRITE must be preceded by a call to READ, and REWRITE updates the record just read by the READ statement. The updated record is named in the FROM phrase and may contain modified alternate keys but not a modified primary key. In dynamic mode, REWRITE does not need to be preceded by READ. The record which gets updated is the first record (in primary key sequence) which has the same primary key value as the updated record passed to REWRITE in the FROM phrase. Again, the updated record is named in the FROM phrase and may contain modified alternate keys but not a modified primary key. REWRITE does not affect the position of the current record pointer.

As you can see from the discussion of WRITE and REWRITE, access must be sequential and not dynamic in order to delete or update records with duplicate primary keys, because in dynamic access only the first record (in primary key sequence) which has a particular primary key value can be deleted or updated.

The COBOL II statements which can be used to lock and unlock indexed files are as follows:

```
EXCLUSIVE file-name [CONDITIONALLY]

UN-EXCLUSIVE file-name
```

Just like with the intrinsics, locking is required for making modifications to a file in shared access. This does not mean, however, that EXCLUSIVE and UN-EXCLUSIVE should just bracket modification operations. If a file is being shared in any way, it should be locked whenever pointer-dependent operations are being executed so that the file does not change unbeknownst to the current record pointer. The locking scheme should be to lock the file before a pointer-independent operation, and to unlock it after all pointer-dependent operations which depend on that pointer positioning have completed.

Note that you do not have to put ";LOCK" on a file equation in order to lock a file with EXCLUSIVE. This is because the COBOL II compiler automatically sees to it that a file is opened with dynamic locking allowed if the EXCLUSIVE statement is used to lock it.

Error handling with COBOL II's Indexed I/O Module can take several different forms. First of all, if no error handling is done, a program will abort with a file information display when an input-output error occurs. Error handling will allow the program to retain control after an input-error rather than automatically aborting.

One way of handling certain errors is by using the INVALID KEY or AT END clauses which can be specified as part of the access and modification statements. If an INVALID KEY or AT END condition arises when a statement is executed, control is transferred to the imperative statement in the INVALID KEY or AT END clause if the statement has such a clause. The situations which cause this to happen are documented for each of the access and modification statements individually in Section XI of the COBOL II manual.

Another input-output error handling technique involves a USE procedure. If a USE procedure is provided for a particular file, control will automatically be transferred to the USE procedure when an input-output error occurs on the file. The only exception to this would be if the error is an INVALID KEY or AT END error and the INVALID KEY or AT END clause was specified on the statement which failed. In that case, control would be transferred to the imperative statement in the INVALID KEY or AT END clause rather than the USE procedure.

The action taken by an INVALID KEY or AT END clause or by a USE procedure will in most cases involve checking the FILE STATUS data area to obtain more detailed information as to what type of error occurred. The meaning of the status information is described in the FILE STATUS clause documentation in Section III of the COBOL II manual. Note that FILE STATUS checking can also be used in lieu of or outside of the INVALID KEY or AT END clauses or USE procedure.

Data Integrity & KSAM Files

There are two basic ways that a KSAM file's integrity can be compromised. First, by a system failure; secondly as a result of improper locking strategy in a multi-user environment. To better understand all the aspects of such problems, it is necessary to briefly review KSAM file structures and look at some KSAM system internals.

KSAM File Structures

KSAM files have two components; a key file and a data file. The key file is maintained by KSAM itself and is not directly accessible by the user. Within the key file, KSAM maintains numerous maintenance pointers as well as modified B-TREE structures for every key field. Whenever a record is added or deleted, these structures must be updated.

The KSAM data file contains maintenance pointers as well as the user data. It also contains a user label in which the name of the key file is stored.

If either one of these files is purged, KSAM will return a "NON-EXISTENT PERMANENT FILE (FSERR 52)" message when trying to access the remaining file.

Both the key and data files are built by the KSAM system using standard MPE file structures. The user specifies the file record structure with the BUILD command in KSAMUTIL which is much like the BUILD command in MPE with regards to record format and extent allocation. KSAM automatically specifies and executes the BUILD command for the key file based on its own internal needs regarding key structures and pointers. Therefore, the MPE file system is responsible for updating file EOFs, allocating new extents, and updating file labels.

KSAM System Internals

KSAM makes use of an extra data segment with which it maintains dynamic information such as internal pointers, key information, and data. As users access the KSAM file, the KSAM system updates the extra data segment to reflect whatever changes have been made to the pointers and data. KSAM determines, based on the number of changes, when to write the updated pointers or data to disc. When the last user closes the KSAM file, all information in the extra data segment is written to disc. Pointers may be written before the data, or data may be written before the pointers. Therefore, at any given time, the linkages between the key and data files on disc may be out of sync pending an update from information in the extra data segment.

Since MPE is responsible for updating file EOFs, KSAM maintains its own internal EOF pointers in the extra data segment for both the key and data files so that it may know whether the amount of information in either file will actually fit on disc. As we will see, KSAM uses this EOF pointer when in recovery mode.

Example:

> Say a program adds several records to a KSAM file. The KSAM software will write those records into its extra data segment. It will also alter the key information placing the new key values in sorted order. At this time, the data is still in memory although KSAM itself is aware of the changes. All users will see the updated chain pointer information.

Let's say that the amount of updated key information in memory will not fit into current size of the key file on disc. KSAM's internal key file EOF pointer then exceeds the physical file EOF as maintained by MPE.

Another transaction occurs and KSAM decides to post the key structures to disc. The file system takes the key information and decides that the key file will have to have another extent allocated to fit the data. MPE increments the key file size, writes the new key information from memory onto disc, and updates its EOF. MPE's file EOF now equals or exceeds the KSAM internal EOF for the key file.

Now, the key information is updated on disc. However the data records associated with the new key values are still in memory and therefore potentially lost if a system failure occurs.

Data Integrity and System Failures

IF a KSAM file is open when a system failure occurs, KSAM will not allow the KSAM file to be used until a KSAMUTIL recover is invoked. Attempts to use it will result in the message "SYSTEM FAILURE OCCURRED WHILE THE KSAM FILE WAS OPENED (FSERR 192)".

KSAM knows the failure has occurred because when it opens its files for access, it saves the system Cold Load ID (a unique number for each system restart). In addition, it keeps a running count of the number of processes which are accessing the file. After a system failure, when an attempt is made to open the KSAM file, KSAM will compare the Cold Load ID in the file to that of the system. If they are different, or if the accessor count is greater than 0, it will not allow the file to be accessed.

There are four types a damage that KSAM file can incur as a result of a failure:

a) Key file information has been written to disc, but data file records have not. In this case, the key file will contain pointers which have no corresponding data records.

   When a KSAMUTIL recover is done, it will delete those key pointers.

b) Data file information has been updated to disc, but the system crashed before MPE could update the physical EOF, and as a result, the KSAMs logical EOF data file pointer is greater than the physical EOF. This means that key file pointers will point to records past the physical EOF.

   When recover is executed, KSAM will set the MPE EOF to equal the KSAM EOF such that no information is lost.

c) Key file information had been written to disc, but the system crashed before MPE could update the physical EOF on the key file. As a result, there are valid key blocks on disc, but past the file's EOF.

   When recover is executed, the KEY file physical EOF is set to KSAMs logical EOF for that file. In this case, no data is lost.

d) The KSAM data file has been updated, but the key file has not. Therefore, data records exist with no corresponding key values.

   KSAMUTIL cannot recover in this situation. It will issue the message, "THERE ARE SOME RECORD(S) WITH KEY VALUE(S) MISSING THE KSAM FILE HAS TO BE RELOADED".

To reload the KSAM file, use FCOPY:

```
                    :RUN FCOPY.PUB.SYS
                    >from=DATAFILE;to=(NEWDATA,NEWKEYS)
```

After reloading the file, KSAMUTIL's purge and rename commands can be used to restore the file to its original name. Do not use the MPE purge and rename commands as they do not logically link the key and data files.

Of course, if a system failure damages the MPE file structure of the data file, the KSAM file will have to be reloaded from tape. Barring this catastrophic occurrence, KSAM offers reliable and safe recovery mechanisms in the event of a system failure.

Data Integrity Problems Due To Locking Strategy

Some of the most common KSAM problems involve data integrity problems that result from using KSAM in a multi-user environment without a proper locking strategy. If locking is not used, or it is not used correctly, data may be overwritten or damaged accidentally.

Locking

Locking can prevent such data integrity problems. When one user locks a file and accesses it, pointers are preserved, chains are maintained, data records changes are known. Then when the user unlocks the file, all the information contained in memory is written to disc so the next user, upon accessing the file, will work with a 'clean' set of data and pointers. Locking can be invoked through the FOPEN intrinsic AOPTIONS, as described in the *MPE File System Reference Manual (P/N 30000-90236)*, or through the use of the MPE FILE command.

When a file has been locked by the user, KSAM will prevent another user from executing any file modifying intrinsics. Once one user opens the KSAM file for dynamic locking, all other users must do so as well. However users need not lock the file to execute the read intrinsics. This can cause potential problems. Consider the following example:

User 'A' does locks the file and read a record. User 'B' does not lock the file, but reads the same record. User 'A' now updates that record with a new value and unlocks the file. User 'B' decides based on the value he has read to update the record, HOWEVER HE DOES NOT REALIZE THAT THE VALUE HAS BEEN CHANGED BY PROCESS 'A'. In this case, process 'A's values will be overwritten. This can be especially dangerous in the case where the values being updated are running totals.

The Worst Case

User 'A' locks the file, reads a record. User 'B' reads the same record. User 'A' now DELETES that record thereby changing the KEY structures and current record pointers for the file. Process 'A' unlocks the file and those changes are posted to disc. Process 'B' now decides to lock the file and delete what it believes to be the SAME record. Because of 'A's updates, process 'B' pointers may now be pointing to a different record than the one it has read. In this case, a DIFFERENT record may be deleted unintentionally!

Correct Locking Strategy

a) Have each user lock the KSAM file when accessing it in a multi-user environment *whether reading or writing*.

b) Make sure that locks occur around logical transactions:

```
FLOCK

FREADBYKEY

FUPDATE/FREMOVE

FUNLOCK
```

This sequence is adequate unless there is a user prompt after the FREADBYKEY. In that case, the KSAM file will be unaccessible to other users while they wait for the locker to decide to update or not. If a user prompt is needed after the read then this is a better locking strategy:

```
FLOCK

FREADBYKEY

FUNLOCK

<<decide to update the data or not>>

FLOCK

FREADBYKEY

FUPDATE/FREMOVE

FUNLOCK
```

Remember, locking must be explicit invoked. Readers should lock, as well as writers, and they should not read a record until they can get exclusive access to the KSAM file.