Chapter 6: File It Under I or O

In which we learn to read and write records.

The term “sequential file” refers to how the records of a file will be processed, and to a lesser extent, the way the file records are physically organized on some media. For input, a sequential file is usually read starting from the first record, proceeding to the second record, then the third, and continuing in this fashion to the end. We will write out the first record for sequential output files, then the second, then the third, proceeding in this manner to the last record. The most commonly used sequential file for IBM mainframes is a QSAM file. QSAM is an acronym for the “queued sequential access method”. In this topic, we investigate how QSAM files are created and processed. We will also learn about record blocking as well as “locate” and “move-mode” input and output.

Defining a QSAM File (Queued Sequential Access Method)

QSAM files are defined inside a program using IBM’s **DCB** macro. This macro generates a block of storage called a “data control block”, which contains information used by the operating system when processing the file. The macro is non-executable and serves only to generate a control block at assembly time. Being non-executable, the macro is coded in the program at a point that would not become part of the execution sequence. Many programmers choose to code the **DCB** just after the program's executable portion and before the variable declarations. This is a reasonably safe location and serves to keep the **DCB**’s from becoming corrupted accidentally by the program. At run time, the information in the **DCB** is combined with information in the data definition statement (DD) of the JCL and information in the data set label to complete the information in the **DCB**. The data set label is a control block created and stored with the file when it is created. Later we will investigate how the information from the **DCB**, the DD statement, and the data set label are combined at run time.

Let’s first look at a sample **DCB** macro as it might appear in a program.

```
CUSTFILE  DCB  DDNAME=CUSTOMER,                             +
                 DSORG=PS,                                  +
                 LRECL=80,                                  +
                 MACRF=(GM),                                +
                 RECFM=FB,                                  +
                 EODAD=ENDFILE
```

By coding a **DCB**, we are defining a file and its characteristics. In the **DCB** above, the file’s internal name, the name used inside the program, is **CUSTFILE**. Whenever the program logic references the file, this is the name that will be used. For instance, to read a record in the file, we might code,

```
GET CUSTFILE,MYREC
```

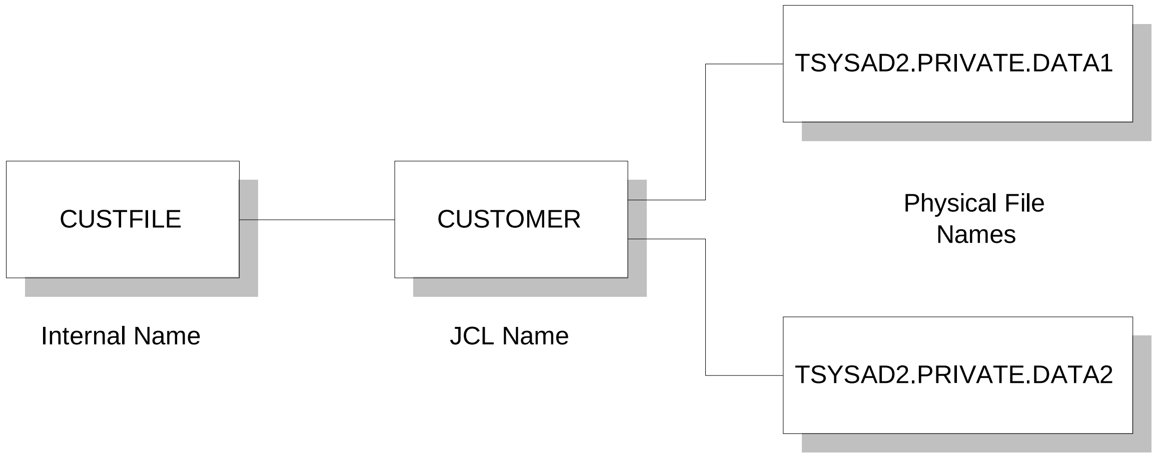
The internal name appears in the first eight columns of the **DCB** macro. This is followed by the word **DCB**. The rest of the macro is a sequence of keyword parameters which may be coded in any order:

1) **DDNAME** - This parameter assigns the file a “second” name, which appears in the DD statement in the JCL used to execute the program. The following is typical of the DD statement that would be part of the JCL,

```
//CUSTOMER DD DSN=TSYSAD2.PRIVATE.DATA,DISP=SHR
```

Notice that the word CUSTOMER, which we will call the “JCL name”, appears in the DD statement's name field and is used to associate the JCL name with a “physical file name”. The physical file name is the actual file name as it might appear in the system catalog. As a result, we have three names for the file we are processing: an internal name, a JCL name, and a physical file name. The purpose of having three names is to provide some “indirection” in the file names so that our program is not tied to a single physical file. By changing the DD statement, we can change the physical file that the program references. We illustrate this below.

```
//CUSTOMER DD DSN=TSYSAD2.PRIVATE.DATA1
//CUSTOMER DD DSN=TSYSAD2.PRIVATE.DATA2
```



Only one of the DD statements above would appear in the JCL. If the first were coded, the connections in the upper path would be indicated. If the second DD were coded, the connections in the lower path would be indicated. As you can see from the diagram, changing the JCL DD statement allows the program to process different physical files by changing the JCL.

2) **DSORG** - This keyword parameter stands for “data set organization” and is used to control the file's basic structure. In this case, PS stands for “physical sequential” and serves to identify the file as a QSAM file. The records in the file are stored and processed sequentially. Records that are “logically” in sequence are often physically stored in sequence on the disk.

3) **LRECL** - The “logical record length” is the number of bytes in the record structure defined by the programmer and processed by the program.

4) **MACRF** - This parameter determines the format for the I/O macros and the mode in which I/O will occur. For QSAM files, there are four possible values: GM, PM, GL, PL. The “G” in the parameter value indicates that a GET macro will be used for accessing the file. This implies the file is an input file and already exists. The “P” means a PUT macro will be used to write records, and the file is for output. The second letter determines the mode in which the I/O will occur. “L” is for locate-mode I/O, and “M” is for move-mode I/O. We will discuss these two modes later.

5) **RECFM** - The RECFM parameter determines whether the program will process fixed-length records (F), or variable-length (V) records. This parameter also controls whether the records are blocked (B) or unblocked. Here are the typical values for this parameter and their meanings.

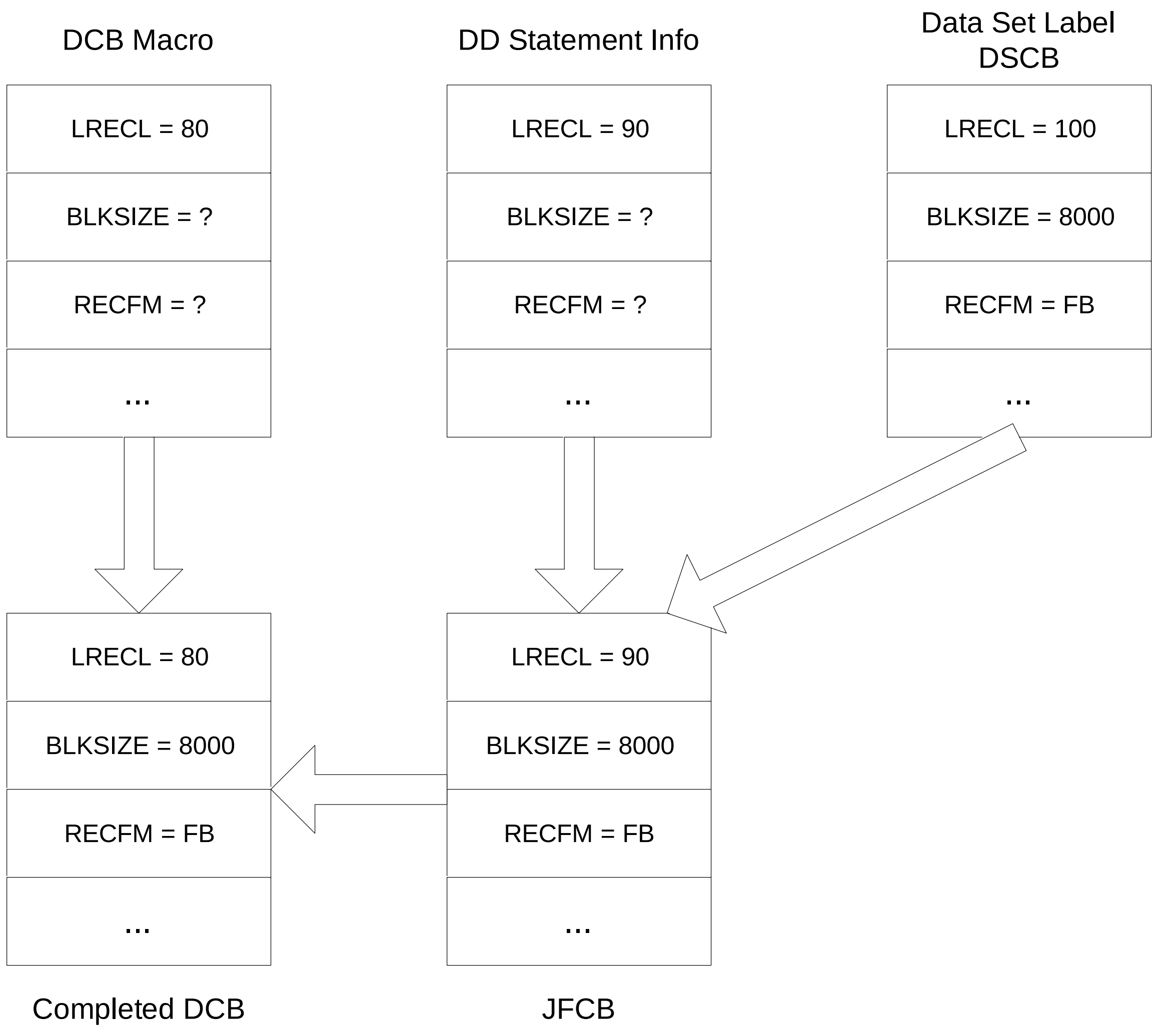
RECFM = F	Fixed size records, unblocked
RECFM = FB	Fixed size records, blocked
RECFM = V	Variable size records, unblocked
RECFM = VB	Variable size records, blocked

We will discuss record blocking later in this topic.

6) **EODAD** - The "End of Data" parameter is only coded for input files (MACRF=GM or GL). This parameter provides a label where the program will branch when the "end of file" condition occurs. The operating system detects the "end of file" condition whenever we invoke a GET on an empty file. At "end of file", the operating system transfers control to the address coded on the EODAD parameter.

Opening a QSAM File

Before you can process records in a file, you must open the file. The open process causes the empty fields in the DCB to be filled so the file can be processed correctly. It helps to understand how information in the DCB is constructed. This process is illustrated in the diagram below. Some of the parameters are impractical but serve to demonstrate how the information is combined.



At assembly time, the DCB is created and initialized with data in the program's DCB macro. When the job is submitted for execution, the JCL is scanned by the Job Scheduler, and a Job File Control Block (JFCB) is created using the information found in the DSCB (if the file already exists). Then the DSCB information is overwritten with any parameters found in the DD statement. When the file is opened at run time, any information missing in the DCB is supplied by the JFCB. Because of the order in which the information is combined, the most important information comes from the parameters coded in the program's DCB macro. This information is supplemented by information gleaned from the DD statement in the JCL. The only parameters taken from the DD statement and stored in the DCB are those missing in the program's DCB. In other words, if a parameter is supplied in the DCB and on the DD statement in the JCL, only the DCB parameter is used. Finally, the only information the data set label contributes is that which was missing in the DCB macro and the DD statement.

The OPEN macro has the following format,

```
OPEN (dcb-address,(processing option))
```

dcb-address - The label in the name field of the DCB macro.

processing option -

INPUT – The data set exists and will be used for retrieving records.
OUTPUT - A new data set is being created, or the records in an existing file will be replaced by the records that will be written by the program.
EXTEND - Records will be added to the end of an existing file.
UPDAT - An existing data set will be used for retrieving records. Additionally, existing records can be modified.

As an example, the following statement opens the file called “CUSTFILE” for input processing.

```
OPEN (CUSTFILE,(INPUT))
```

CLOSING A QSAM FILE

After we process a file, it should be closed. This process logically disconnects the program from the file. During the close processing, the program DCB is reconfigured with the parameters it initially contained at assembly time. This means the file can be opened again for further processing.

The format of the CLOSE statement follows below,

```
CLOSE (dcb-address-1,dcb-address-2,...)
```

dcb-address-n - The label in the name field of the DCB macro.

The two statements below illustrate how to code a CLOSE statement.

```
CLOSE (CUSTFILE)
CLOSE (CUSTFILE,MASTFILE)
```

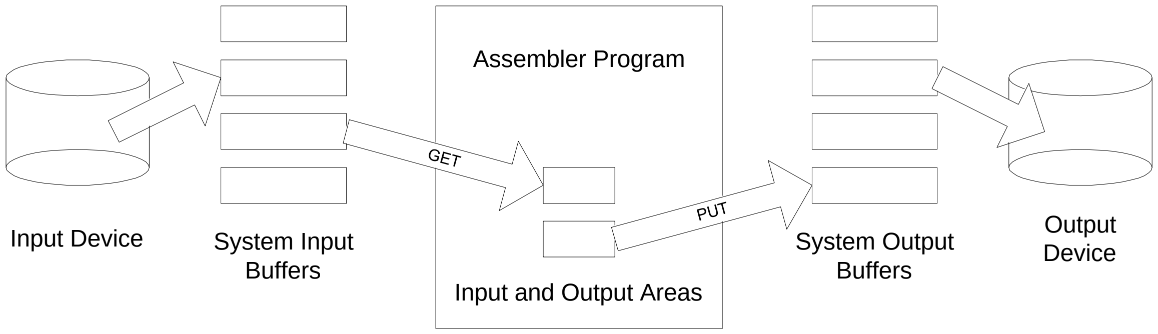
In the first CLOSE statement, the CUSTFILE is closed, while the second CLOSE statement closes two files with one statement.

Record Queuing

The “Q” in QSAM stands for “queued”, and refers to the queuing of records during input and output processing. On the input side, records are brought from an external source (disk, tape, etc.) into the machine's main memory. The records are delivered into storage areas called “system buffers” where they reside until retrieved by the program using a GET macro. The queuing process begins when the program opens the file. The operating system starts delivering records to the system buffers even before execution of the first GET. During program execution, the operating system tries to keep the system buffers full so that the program will not wait for records to be retrieved from the external device. This has the effect of speeding program execution.

During output processing, PUT macros place records into system buffers where they reside until the operating system can retrieve them and transfer them to an external storage device. By "buffering" the output, the program can continue execution without waiting on the external device.

Input and Output queuing is illustrated in the diagram below.



As depicted above, the diagram illustrates "move-mode" input and output. The term "move-mode" refers to the process of moving a record from a system input buffer to a program input area or from a program output area to a system output buffer. These moves occur as a result of coding GET and PUT. Locate-mode I/O involves processing records in system buffers rather than user areas defined inside a program.

Reading and Writing QSAM Records in Move-mode

After opening a file for input, the records are available for retrieval. To read a record, you must code a GET macro. There are two formats for this macro, depending on how the MACRF parameter is coded in the file's DCB macro. Coding "MACRF=GM" determines that records will be retrieved in "move-mode". When we read a record, a copy is delivered to a storage area defined in the program. This storage area is called a "buffer" and reflects the contents of the record. Here is an example.

```
          GET  CUSTFILE,CUSTREC
          ...
CUSTREC   DS  0CL80
CUSTNAME  DS  CL40
CUSTINF1  DS  CL20
CUSTINF2  DS  CL20
```

The GET macro above names the file as its first parameter and the program buffer area as its second parameter. Since we are assuming move-mode input, a record is delivered from a system buffer to the storage area called CUSTREC.

For output processing, use the PUT macro to write records to a file. The MACRF parameter determines the mode in which we process records. MACRF=(PM) indicates move-mode output. An example PUT macro is listed below.

```
         PUT  MASTFILE,MASTREC
         ...
MASTREC  DS  0CL100
```

```
MASTID   DS  CL8
         ...
```

First, the record is created in a program area called MASTREC. All the fields of the record would be initialized with appropriate values. When the record is complete, it is written to the file by executing the PUT macro. The first parameter in the macro names the output file DCB. The second parameter names the buffer containing the record. Executing the macro causes the information in MASTREC to be transferred to a system buffer where it will be processed later.

Reading and Writing Records in Locate-Mode

When MACRF=(GL) is coded, input processing will occur in "locate-mode". Processing in this mode is more efficient than in move-mode since the records we read are never transferred directly to the program's storage area but instead are left in the system buffers. For files with large numbers of records or large record sizes, the processing time saved using locate-mode rather than move-mode can be substantial. If the records are not transferred directly to a program buffer, how can the program access the information in a record? The answer is that the programmer must use a DSECT to reference the storage. (See **Assembler for Dummies**.) The GET macro takes an alternate form for locate-mode processing. In this format, the macro has a single parameter, which is the file DCB name. A sample locate-mode GET is coded below.

```
GET CUSTFILE
```

After executing the macro, the operating system initializes register 1 with the address of the record delivered as a result of the GET. This address will be pointing inside a system buffer. Providing access is a simple matter of loading the record's address into the register associated with the DSECT. This is illustrated below.

```
CUSTREC     DSECT
CUSTBAL     DS  PL4            CUSTOMER BALANCE
            ...                OTHER DSECT FIELDS

            USING CUSTREC,R5
            GET  CUSTFILE
            LR  R5,R1          MAKE R5 POINT AT THE RECORD
            ZAP  TEMP,CUSTBAL  PROCESS THE FIELDS IN THE RECORD
            ...
```

After the GET is executed, the address of the delivered record is placed in register one. Subsequently, the address is loaded into register 5 by the **LR** instruction. The USING statement provides the association between the DSECT name and register 5. With register 5 loaded with the appropriate address, addressability to the record is established with the names in the DSECT.

Locate-mode output is indicated by coding MACRF=(PL) in the DCB macro. The PUT macro is executed to write a record to a file.

```
PUT  MASTFILE
```

Executing the PUT causes the operating system to place the address of an available record area inside a buffer in register one. Using the **LR** instruction, this address is copied to a register that controls an output DSECT. Once addressability has been established to the output record, the program creates the record by moving data to the record. The record remains available for further program processing until the next PUT is issued or the file is closed. The following code is typical of locate-mode output.

```
MASTREC     DSECT
MASTNO      DS   CL5              CUSTOMER NUMBER
MASTBAL     DS   PL5              CUSTOMER BALANCE
            ...
            USING MASTREC,R6
            PUT  MASTREC
            LR  R6,R1             MAKE R6 POINT AT EMPTY REC
            ZAP  MASTBAL,BALPK    REC FIELDS AVAILABLE
            ...
```

Keep in mind that register 1 is a "volatile" register and is subject to change when executing a system macro or calling another program. Be sure to make a copy of register 1 **immediately** after executing PUT or GET, and certainly before issuing another system macro.

Record Blocking and Deblocking

Blocking refers to the operating system process of combining multiple logical records into larger physical records called "blocks". A **logical record** is the record structure defined by the programmer and consists of a collection of related fields that logically belong together. A **physical record** is a collection of logical records that have been combined to store them efficiently on an external device like a disk or tape drive. Records are blocked because the process of accessing externally stored data is expensive in terms of CPU time. For instance, in the time it takes to move a disk arm, hundreds of thousands of instructions can be executed by the CPU. For efficiency, rather than returning a single record when a program requests a "read" operation, the operating system delivers an entire block of records from disk to memory. The process of separating records from a block and delivering them individually to a program is called **deblocking**.

The RECFM parameter determines whether records will be blocked or not. Choosing RECFM =FB or VB selects the blocked format. In practice, most files are blocked. The exception is made for files with records containing thousands of bytes. In the pictures above, the system buffers correspond to blocks. Individual records are delivered to the program either in move or locate-mode.

The programmer can also control the block size with the BLKSIZE parameter that is coded in the DCB. For example, if the programmer has coded LRECL=80 and BLKSIZE=8000, then each block will contain 100 logical records. Computing an optimal block size by hand requires detailed knowledge of the device on which the data is recorded. For IBM's latest operating systems, block sizes are computed automatically if the BLKSIZE parameter is omitted in the DCB and on the DD statement when the file is created.

Carriage Control for Printed Reports

If an output file is to be physically printed as a report, you'll need to add a one-byte carriage control character to the front of each output record to indicate what the printer is supposed to do with the record (write and skip one line, write and skip two lines, skip to the top of a page, etc., …).

There are two flavors of carriage control characters: 1) IBM and 2) ASA (American Standards Association). The main difference involves the timing of writing a record and advancing the carriage. With IBM carriage control characters, first the record is written and then the carriage is advanced. With ASA carriage control characters, the carriage is advanced and then the record is written. In COBOL, this corresponds to `WRITE … BEFORE ADVANCING` and `WRITE … AFTER ADVANCING`.

You can announce your intention to add carriage control characters to each record by appending an "A" (for ASA) or an "M" (for IBM) to RECFM parameter of the output file. For example, RECFM=FBM, would be used to add IBM carriage control characters to each record. The LRECL length of the file will also have to be increased by 1 to accommodate the carriage control character.

Before writing each record, add the carriage control to the first byte of the output buffer using MVI. You will need to define the carriage control characters you are using in separate fields like this,

```
*********
*  IBM CARRIAGE CONTROL CHARACTERS
*********
WSPC0    EQU   X'01'      OVERSTRIKE
WSPC1    EQU   X'09'      WRITE, SPACE 1 LINE
WSPC2    EQU   X'11'      WRITE, SPACE 2 LINES
WSPC3    EQU   X'19'      WRITE, SPACE 3 LINES
TOPPAGE  EQU   X'8B'      TOP OF PAGE (CHANNEL 1) NO WRITE
*********
*  ASA CARRIAGE CONTROL CHARACTERS
*********
SPC1W    EQU   C' '       SPACE 1 LINE THEN WRITE
SPC2W    EQU   C'0'       SPACE 2 LINES THEN WRITE
SPC3W    EQU   C'-'       SPACE 3 LINES THEN WRITE
OVERSTK  EQU   C'+'       OVERSTRIKE
NEXTPG   EQU   C'1'       NEXT PAGE THEN WRITE
*********
```

Printing a heading at the top of a page with IBM carriage controls would involve two PUT operations. The first PUT with carriage control x'8B', would skip to the top of a page without writing. The second PUT, with carriage control x'11', would write the heading and advance 2 lines. Subsequent PUTs for detail lines would use x'09' to write and space 1 line.

PROBLEMS

6-1. What do the letters in DCB stand for?
6-2. Is a DCB an executable macro?
6-3. What is the purpose of a DCB?
6-4. What are the three sources of information that are used to construct a DCB?
6-5. Why is the length of a record in a DCB specified as a “logical record length”?
6-6. When do you need to code an EODAD parameter in a DCB?
6-7. What does the MACRF parameter control?
6-8. What are the four most common values of the RECFM parameter?
6-9. What is the main difference between Move-mode and Locate-mode I/O?
6-10. Which QSAM I/O mode is most efficient?
6-11. Why are records blocked?
6-12. What is record queueing?

Programming Exercise

1. Take the sample program in chapter 3 that uses Move-mode QSAM for output and convert to print a report. Add a header at the top of the page. Double-space the header from the detail lines. Single-space the detail lines. If you look at the output in SDSF, you will need to issue SET HEX ON to see the carriage control characters.