6. QUEUED SEQUENTIAL ACCESS METHOD

6.1 What is Queued Sequential Access Method?

Queued Sequential Access Method, or QSAM, is An **access method** is a complete set of macros and modules that are used to perform input/output operations.

QSAM is used to process sequential files and is considered one of the most straightforward of the access methods because it does a lot of the work for a programmer.

A data set is read into a buffer in memory one **physical record** or **block** at a time. One physical record is made up of a fixed number of **logical records**. The fixed number is known as the **blocking factor**.

QSAM provides two services that some other access methods do not:

1. buffering
2. deblocking

Buffering is the reading of one block at a time into a buffer in memory.

Deblocking is the dividing of the physical record into the logical records.

QSAM is implemented (in ASSEMBLER) by the use of 5 macros:

1. `DCB`
2. `OPEN`
3. `CLOSE`
4. `GET`
5. `PUT`

6.2 DCB Macro

The data control block, or DCB, macro is used to describe each of the data set(s) that will be read from or written to.

The format is:

```
dcbname  DCB  DDNAME=ddname,                                    X
              DEVD=_____,                                       X
              DSORG=_____,                                      X
              MACRF=_____,                                      X
              RECFM=_____,                                      X
              LRECL=_____,                                      X
              BLKSIZE=_____,                                    X
              EODAD=_____
```

- `dcbname`

  The name that will be referenced in the source code when opening or closing a file.

- `DDNAME=ddname parameter`

  The keyword parameter DDNAME links the DCB to the actual file being read from and/or written to and the ddname corresponds to the file's DD statement in the JCL.

- `DEVD=_____ parameter`

  The keyword parameter device drive specifies the type of device on which the file is stored. There are three possible values:

  `DA` direct access (always use this one)
  `UT` utility class (tape or disk)
  `UR` printer

- `DSORG=_____ parameter`

  The keyword parameter data set organization specifies whether the data set is physical sequential, partitioned organization or other. Most of the time PS is used. It represents physical sequential even though the actual data set is a member of a PDS or PDSE.

- `MACRF=_____ parameter`

  The keyword parameter macro format specifies which format of the GET and PUT macros will be used. There are four possible values:

GL Get Locate

The address of the next logical record to be read is placed in register 1 and the data is not actually copied into the program but you instead have a reference to it in register 1.

GM Get Move

The logical record is copied into a named area of storage.

PL Put Locate

The address of the next available buffer area for writing is returned in register 1. The program must copy the logical record to the address referenced in register 1.

PM Put Move

A logical record is written from a named area of storage.

If a file will be used for both input and output, specify `(G_,P_)` as the macro format.

- `RECFM=_____ parameter`

  The keyword parameter record format specifies what type of records are being written or read. There are three types:

  `F` fixed length records
  `V` variable length records
  `U` undefined length

  Fixed and variable can be further subdivided using:

  `B` blocked `(FB or VB)`
  `A` ASA carriage control character in 1st byte `(FA or VA)`
  `M` Machine carriage control character in 1st byte `(FM or VM)`

- `LRECL=_____ parameter`

  The keyword parameter logical record length specifies the number of bytes in a logical record.

- `BLKSIZE=_____ parameter`

  The keyword parameter block size is an integer value that specifies the number of bytes in a block, or physical record. It must be an even multiple of the `LRECL`. It is not necessary to code `BLKSIZE` if the data set being read from or written to already exists.

- `EODAD=_____ parameter`

  The keyword parameter end-of-data address specifies the name of an end-of-file routine that will be execute after all of the input records have been read. The last instruction of an `EODAD` routine is usually

branch register to the address in register 14 as register 14 holds the address of the instruction following the `GET` that attempted to read another record but there are no more.

6.3 OPEN macro

The OPEN macro is used to open a file for input or output. To open a file, or DCB, for input:

```
OPEN  (dcbname,(INPUT))
```

To open a file, or DCB, for output:

```
OPEN  (dcbname,(OUTPUT))
```

To open more than one file at a time:

```
OPEN (dcbname1,(INPUT[OUTPUT]),dcbname2,(INPUT[OUTPUT]),...)
```

The OPEN macro sets a return code in register 15. A return code of 0 indicates success and any other value indicates failure.

If INPUT or OUTPUT are not coded, INPUT is assumed but it must be coded with the comma following the dcbname:

```
OPEN  (dcbname,)
```

6.4 GET macro

This macro is used to read a single logical record from a file.

As stated above in section 6.2, the DCB keyword parameter MACRF, or macro format, specifies which form of the GET and PUT macros are being used to read or write data for the file associated with the DCB.

GM Get Move

```
        GET   dcbname,buffername
```

A single logical record is read from the file associated with the DCB named dcbname and placed into a user-defined storage area named buffername. For example:

```
        GET   SALESDCB,SALESREC
```

This example will read a logical record from the file associated with the DCB named SALESDCB and copy it into a buffer named SALESREC in the programs storage. If the file has LRECL=80, that storage area could be defined as:

```
SALESREC DC    80C' '
```

GL Get Locate

```
        GET    dcbname
```

A single logical record is read from the file associated with the DCB named dcbname and the address of that record in storage is placed in register 1. For example:

```
        GET   SALESDCB
```

This example will read a logical record from the file associated with the DCB named SALESDCB and returns the address of the record in register 1. To copy it into a buffer named SALESREC in the programs storage, the GET must be followed by a move character, or MVC. For example, if the file has LRECL=80, the following will read the record and the MVC following the GET will copy it into the user-defined storage buffer named SALESREC:

```
        GET   SALESDCB
        MVC   SALESREC(80),0(1)
```

If a GET is executed and there are no more records in the file, control is passed by the DCB to the routine specified by the EODAD keyword parameter in the DCB definition itself. Refer to the the explanation of the EODAD keyword parameter in section **6.2 DCB Macro** above.

Each time you issue an OPEN, GET, PUT, or CLOSE operation in QSAM, a file status code is returned in register 15. The most useful of those are:

Value	Meaning
00	Successful completion of any I/O operation.
10	End-of-file detected on a Read, as in At End.

35	Attempt to Open a file not present.
39	Open failure due to a conflict in DCB data.
41	Open failure; file already open.
42	Close failure; file not open.
46	Read failure; last Read already reached EOF.
47	Read failure; file not open for Input.
48	Write failure; file not open for Output.

6.5 PUT macro

This macro is used to write a single logical record to a file or standard output.

As stated above in section 6.2, the DCB keyword parameter MACRF, or macro format, specifies which form of the GET and PUT macros are being used to read or write data for the file associated with the DCB.

PM Put Move

```
         PUT   dcbname,buffername
```

A single logical record is written to the file associated with the DCB named dcbname from a user-defined storage area named buffername. For example:

```
         PUT   REPRTDCB,DETAIL1
```

This example will write a logical record to the file or standard output associated with the DCB named REPRTDCB and the data written will be copied from the storage area named DETAIL1. If the file has LRECL=133, that storage area could be defined as:

```
DETAIL1  DC    C'0'            CARRIAGE CONTROL FOR DOUBLE SPACING
BRKRNME  DS    CL25
         DC    10C' '
SALESAMT DS    CL15
         DC    82C' '
```

Of course, the fields BRKRNME and SALESAMT would have to be moved into the storage definition named DETAIL1 BEFORE the PUT is executed.

PL Put Locate

```
         PUT    dcbname
```

A single logical record is written to the file associated with the DCB named dcbname and the address of that record in storage is placed in register 1. For example:

```
         PUT   REPRTDCB
```

This example will write a logical record to the file associated with the DCB named SALESDCB and returns the address of the record in register 1. To actually copy the output data defined as DETAIL1 in the program's storage out to the file, the PUT must be followed by a move character, or MVC. For example, if the file has LRECL=133, the following will write the record and the MVC following the PUT will copy it to the file from the user-defined storage buffer named DETAIL1:

```
         PUT   REPRTDCB
         MVC   0(133,1),DETAIL1    COPY THE 133 BYTES FROM DETAIL1 TO
*                                    0 BYTES OFF THE ADDRESS IN R1
```

6.6 CLOSE macro

This macro is used to close a DCB associated with a file. The format is:

```
CLOSE (dcbname)
```

It is also possible to close more than 1 DCB at a time with:

```
CLOSE (dcbname1,,dcbname2,,...)
```

As with OPEN, the CLOSE sets a return code in register 15 where 0 indicates success and anything other than 0 indicates a failure to close the file properly.

6.7 QSAM in COBOL

QSAM is also the access method that is commonly used in COBOL behind the scenes. The DCB is supplied information from the SELECT statement, the FD, or File Definition, and some PROCEDURE DIVISION statements. For example:

In the ENVIRONMENT DIVISION, we have

```
           SELECT INFILE ASSIGN TO S-SYSIN.
```

In the DATA DIVISION, we have

```
       FD  INFILE
           RECORDING MODE IS F.

       01  IN-REC.
           05  FIRST-HALF    PIC X(40).
           05  SECOND-HALF   PIC X(40).
```

And in the PROCEDURE DIVISION, we have

```
           OPEN INPUT INFILE.

           READ INFILE AT END MOVE 'Y' TO EOF-FLAG.
```

This COBOL code will produce the following DCB:

```
INFILE   DCB   DDNAME=SYSIN,                       X
               DSORG=PS,                           X
               LRECL=80,                           X
               MACRF=GL,                           X
               RECFM=FB,                           X
               EODAD=something
```

The EODAD parameter will have the address of wherever the AT END part of the READ statement indicates that needs to happen, such as MOVE 'Y' TO EOF-FLAG in the following:

```
           READ INFILE INTO WS-RECORD
             AT END MOVE 'Y' TO EOF-FLAG
           END-READ.
```

A READ INTO like above will change the default MACRF=GL to GM.

```
           WRITE PRINT-LINE.
```

This WRITE will use a MACRF=PL but

```
           WRITE PRINT-LINE FROM WS-RECORD.
```

will change it to MACRF=PM.

When a COBOL program is executed and a file is OPENed, the DCB information is collected from, in order:

1) inside the COBOL program itself
2) DD statements in the JCL
3) Label records (if any) for the data set being opened

This process is known as the forward merge although it is far more complicated than the three steps above.

6.8 Example QSAM Program

```
QSAMPGM  CSECT
*
         PRINT NOGEN
*
         XSAVE BR=12,SA=MAINSAVE,TR=NO
*
         OPEN  (OUTDCB,(OUTPUT))       OPEN THE OUTPUT DCB
         LTR   15,15                   TEST FOR SUCCESS
         BZ    OPEN1OK                 BRANCH PAST ABEND IF SUCCESSFUL
         ABEND 777,DUMP                ABEND AND DUMP IF NOT
*
OPEN1OK  OPEN  (INDCB,(INPUT))         OPEN THE INPUT DCB
         LTR   15,15                   TEST FOR SUCCESS
         BZ    OPEN2OK                 BRANCH PAST ABEND IF SUCCESSFUL
         ABEND 888,DUMP                ABEND AND DUMP IF NOT
*
OPEN2OK  GET   INDCB,INREC             READ FIRST RECORD
*
LOOP1    CLI   EOFFLAG,C'Y'            CHECK FOR END-OF-FILE
         BE    ENDLOOP1                BRANCH TO END IF EOF
*
         MVC   OUTNAME(15),INREC       COPY THE NAME TO OUTPUT RECORD
*
         PUT   OUTDCB                  WRITE THE OUTPUT RECORD
         MVC   0(133,1),OUTREC         MOVE IT INTO THE OUTPUT BUFFER
*
         GET   INDCB,INREC             READ THE NEXT RECORD
         B     LOOP
*
ENDLOOP1 CLOSE (INDCB,,OUTDCB)         CLOSE INPUT AND OUTPUT DCBs
*
         XRETURN RC=0,TR=NO
*
         LTORG
*
*  THE FOLLOWING TWO LINES WILL BEGIN YOUR STORAGE ON A 32-BYTE
*  BOUNDARY MAKING IT EASIER TO FIND, ESPECIALLY SINCE IT WILL BEGIN
*  WITH THE ‘STORAGE FOR QSAMPGM’ LABEL.
*
         ORG   QSAMPGM+((*-QSAMPGM+31)/32)*32
         DC    CL32'***** STORAGE FOR QSAMPGM ******'
*
MAINSAVE DC    18F'-1'                 STORAGE SAVE AREA
*
INREC    DC    80C' '                  INPUT RECORD BUFFER
*
OUTREC   DC    C'0'                    DOUBLE SPACING CARRIAGE CONTROL
OUTNAME  DS    15C                     OUT NAME
         DC    117C' '
```

```
*
*  INFILE EXISTS SO THE PARAMETERS LIKE LRECL AND BLKSIZE CAN BE
*  GOTTEN FROM THE DATA SET'S LABEL RECORDS BUT IT CAN ADDED ANYHOW
*
INDCB    DCB   DDNAME=INFILE,                                    X
               DEVD=DA,                                          X
               DSORG=PS,                                         X
               MACRF=GM,                                         X
               LRECL=80,                                         X
               EODAD=EOFRTN
*
EOFRTN   MVI   EOFFLAG,C'Y'           CHANGE EOF FLAG TO YES
         BR    14                     RETURN TO STATEMENT AFTER GET
*
EOFFLAG  DC    C'N'                   EOF FLAG FOR INFILE.
*
*
*  OUTFILE IS A NEW FILE SO ALL PARAMETERS ARE PRESENT
*
OUTDCB   DCB   DDNAME=OUTFILE,                                  X
               DEVD=DA,                                          X
               DSORG=PS,                                         X
               LRECL=133,                                        X
               BLKSIZE=1330,                                     X
               RECFM=FBA,                                        X
               MACRF=PL
*
         END   QSAMPGM
```

6.9 More About QSAM

The term "sequential file" refers to the manner in which the records of a file will be processed, and to a lesser extent, the way in which 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. For sequential output files, we will write out the first record, 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 "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

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 that is used by the operating system when processing the file that the macro defines. 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 which would not become part of the execution sequence. Many programmers choose to code the DCB just after the executable portion of the program, and before the declarations for variables. This is a fairly safe location and serves to keep the DCB's from becoming corrupted by the program accidentally. At run time, the information in the DCB is combined with information in the job control language data definition statement (DD), as well as information in the data set label in order to complete the information that is stored in the DCB. The data set label is a control block that is 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.

Lets first look at a sample DCB macro as it might appear in a program.

```
CUSTFILE DCB   DDNAME=CUSTOMER,                          X
                      DSORG=PS,                          X
                      LRECL=80,                          X
                      MACRF=GM,                          X
                      RECFM=FB,                          X
                      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 that will be used inside the program, is "CUSTFILE". Whenever the program 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 8 columns of the macro. This is followed by the macro name, "DCB". The rest of the macro is a sequence of keyword parameters which may be coded in any order:

DDNAME

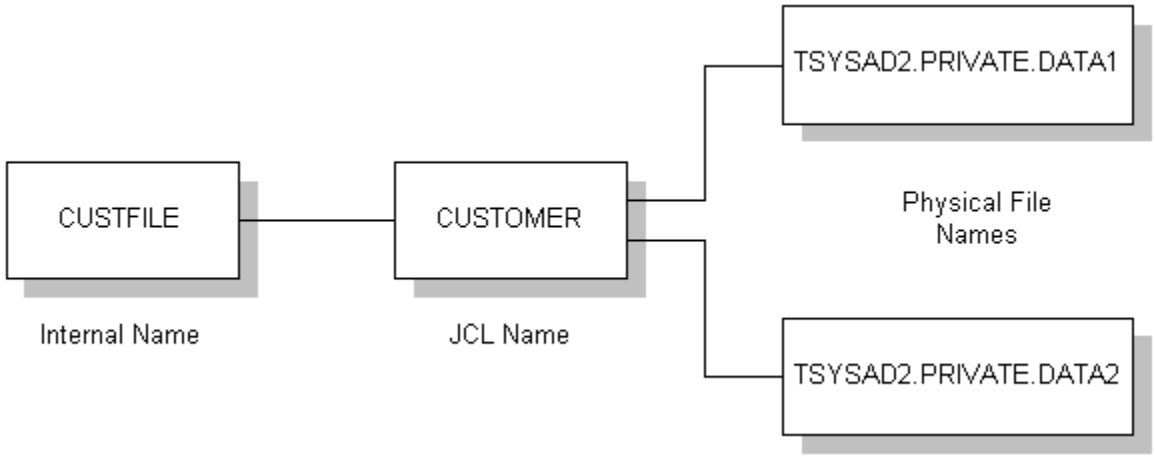
This parameter assigns the file a "second" name which appears in the DD statement in the JCL that is used to execute the program. The following is typical of the DD statement that would be part of the JCL.

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

Notice that the word "CUSTOMER", which we will call the "JCL name", appears in the name field of the DD statement 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 for 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. This is illustrated 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 was coded, the connections in the upper path would be indicated. If the second DD was 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 easily process different physical files.

DSORG

This keyword parameter stands for "data set organization" and is used to control the basic structure of the file. In this case, PS means "physical sequential" and serves to identify the file as a QSAM file. The records in the file are stored and processed sequentially. Records which are "logically" in sequence are also physically stored in sequence on the disk.

LRECL

The "logical record length" is the number of bytes in the record structure defined by the programmer and processed by the program.

MACRF

This parameter determines the format for the I/O macros that will be used to process the file and the mode in which the I/O will occur. For QSAM files there are four possible values that can be coded: GM, PM, GL, PL. The "G" in the parameter value means that the GET macro will be used for accessing the file. This implies the file is an input file and already exists. The "P" means the PUT macro will be used to process the file. In this case the file is an output file. The second letter indicates the mode in which the I/O will occur. "L" means locate mode I/O and "M" means move mode I/O. These two modes will be discussed later in this topic.

RECFM

The RECFM parameter determines whether the logical records the program processes are fixed in length (F), or of variable lengths (V). 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

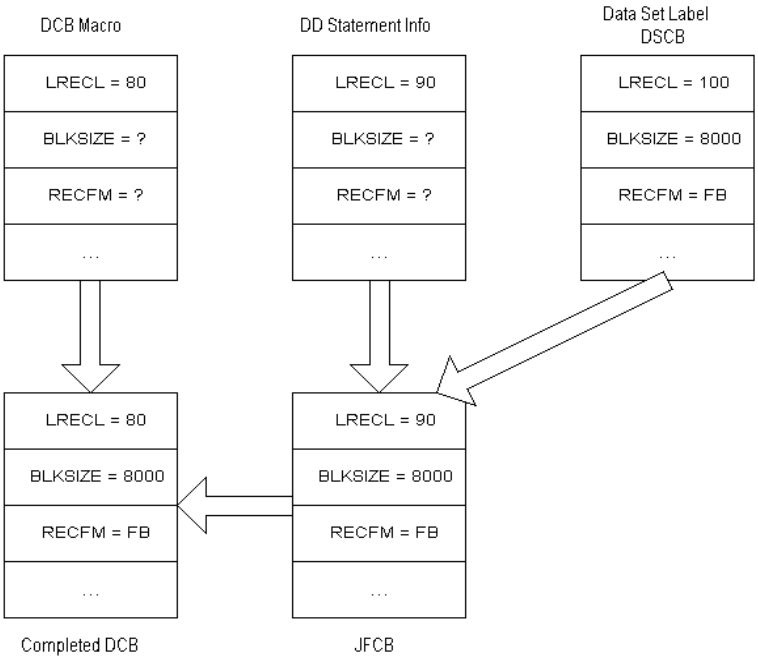
The concept of record blocking will be discussed later in this topic.

EODAD

The "End of Data" parameter is only coded for input files (MACRF=GM or GL). This parameter provides a label to which the program will automatically branch when the "end of file" condition occurs. The operating system detects the "end of file" condition while executing a GET macro when there are no records in the file left to be read. Upon detecting "end of file", the operating system transfers control to address coded on the EODAD parameter.

Opening a QSAM File

Before you can process any records in a file, the file must be "opened". The open process causes the "empty" fields in the DCB to be filled in so the file can be processed correctly. It is helpful 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 illustrate how the information is combined.



At assembly time, the DCB is created and initialized with information contained in the program's DCB macro. When the job is submitted for execution, the JCL is initially scanned by the Job Scheduler, and a

Job File Control Block (JFCB) is created first using the information found in the DSCB if the file already exists, and then overwritten with any parameters found in the DD statement. The file is opened at run time, and any information that is missing in the DCB is supplied by the information in the JFCB. Because of the order in which the information is combined from the DCB macro, the DD statement, and the DSCB, the most important information is that which is placed in the DCB as a result of 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 information that is taken from the DD statement and stored in the DCB, are those fields which were not supplied in the program 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 that is gathered from the data set label is that which was not found in the DCB macro nor in 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

An existing data set is to 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 is to be used for retrieving records. Additionally, existing records can be modified.

For example, the following statement opens the file called "CUSTFILE" for input processing.

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

CLOSING A QSAM FILE

After a file has been processed, 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 are examples of how the CLOSE statement can be coded.

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

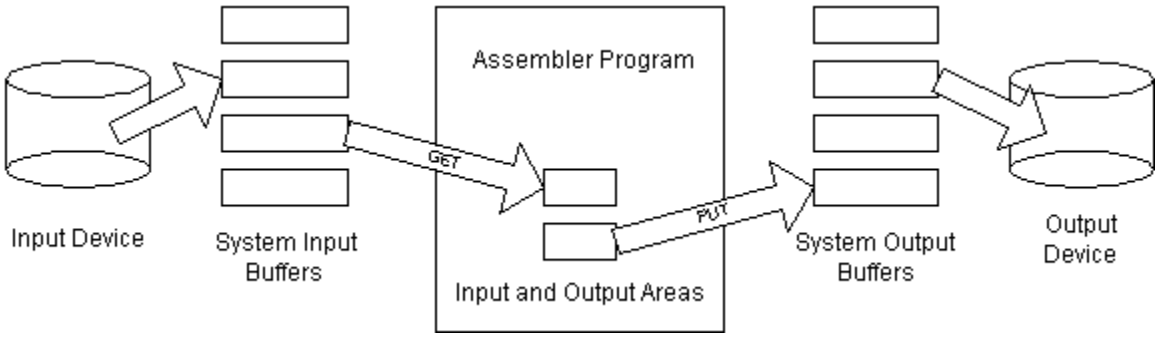
In the first CLOSE statement, the CUSTFILE is closed while the second CLOSE statement closes two files with one statement.

Record Queuing

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

During output processing, records that are produced by the program using the PUT macro are also placed 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.

Reading and Writing QSAM Records in Move Mode

After a file has been opened 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". This means that when a record is read, a copy of it will be delivered to a storage area defined by

the programmer in the program. This storage area is called a "buffer" and is designed to reflect 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, the PUT macro is used for writing records to a file. The MACRF parameter determines the mode in which records will be processed with MACRF=(PM) indicating move mode input and output. An example PUT macro is listed below.

```
         PUT   MASTFILE,MASTREC
         ...
MASTREC  DS    0CL100
MASTID   DS    CL8
         ...
```

First the record that is to be recorded on the file is created in a program area called MASTREC. All the fields of the record would be initialized with appropriate values. When the record has been constructed, it is written to the file by executing the PUT macro. The first parameter in the macro is the DCB name of the output file. The second parameter is the buffer containing the record. Execution of the macro causes the information in MASTREC to be transferred to a system buffer where it will be processed at a later time.

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 that is saved by 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 **DSECTS**.) 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 one with the address of the record that was delivered as a result of the GET. This address will be inside a system buffer. Providing access is a simple matter of loading the address of the record into the register which is associated with the DSECT. This is illustrated below.

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

```
         USING $CUSTREC,5
         GET   CUSTFILE
         LR    5,1                  MAKE R5 POINT AT THE RECORD
         ZAP   TEMP(4),$CUSTBAL(4)  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 6. With register 5 loaded with the appropriate address, addressablility to the record is established with the names in the DSECT.

Locate mode output is indicated by coding MACRF=PL in the DCB macro. To write a record to a file, the PUT macro is executed first.

```
         PUT   MASTFILE
```

Executing the PUT causes the operating system to place the address of an available buffer in register one. Using the **LR** instruction, this address is then copied to a register that controls an output DSECT. Once addressability has been established to the output record, the record is created by the program. 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,6
         PUT   $MASTREC
         LR    6,1     MAKE R6 POINT AT EMPTY REC
         ZAP   $MASTBAL(5),BALPK(5)   REC FIELDS AVAILABLE
         ...
```

Keep in mind that register one 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.

Record Blocking and Deblocking

The term **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 which have been combined for the purpose of storing 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 choice of blocking or not blocking a group of records is made by the programmer when coding the RECFM parameter. Choosing RECFM=FB or VB selects the blocked format. In practice, most files are blocked. The exception is made for files with large records containing thousands of bytes. In the pictures above, the system buffers correspond to blocks from which individual records are delivered to the program either in move or locate mode.

The programmer can also control the size of the blocks in a file using the BLKSIZE parameter coded in the program's 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 requires a knowledge of the device on which the data is recorded and is beyond the scope of this discussion. For IBM's ESA operating system, block sizes will be computed automatically if the BLKSIZE parameter is omitted in the DCB and on the DD statement when the file is created.

This document is courtesy of

http://csc.columbusstate.edu/woolbright/QSAM.HTM