The REXX Queue exists as a part of IBM's support for the REXX environment on the AS400/iSeries/Power Systems OS. The REXX external data queue provides a method to hold temporary data which REXX or a HLL program may use. The data on the queue is accessible by and visible to users as lines, or as buffers. Each line may contain up to 32,767 characters. Individual characters have no special meaning to REXX, so special characters or attributes have no effect on REXX.

The REXX data queue comes into existence when a job is started. There is no special command or consideration for the creation of the queue. It simply begins as a part of the job initiation. The queue persists until the end of the job. Data may be placed in the queue by REXX or user programs in an arbitrary manner. All programs running under the same job have access to the queue so it can be used to provide a data exchange method between programs, offering high-speed inter-program communications.

Here is a fixed format example of code to push data to the REXX Queue. The subroutine in Fig.1 is designed to put an entry on the REXX queue. The '1' in the flag parameter designates that the entry will be placed in the queue for LIFO processing. A zero in conjunction with the 'A' would tell the API the stack will be processed in FIFO (First-In-First-Out) sequence.

```
      * This is the data structure necessary for the binary fields
      * used as parameters on the QREXQ API.
      *----------------------------------------------------------------
     D BFLDS           DS
     D BUFLEN                         9B 0
     D FLAG                           4B 0
     D RCODE                          4B 0
      *---------------------------------------------------------------------
      * BEGIN of work fields
      *---------------------------------------------------------------------
     D BUF             S            256
     D FUNT            S              1
.
.
.

      *================================================================
      * @PUSH - ADD TO STACK
      *================================================================
     C     @PUSH         BEGSR
     C                   MOVEL     COMAND        BUF
     C                   MOVE      'A'           FUNT
     C                   EVAL      BUFLEN = 264
     C                   EVAL      FLAG = 1                              *Lifo
     C                   EVAL      RCODE = *ZERO
      *----------------------------------------------------------------
     C                   CALL      'QREXQ'
     C                   PARM                    FUNT
     C                   PARM                    BUF
     C                   PARM                    BUFLEN
     C                   PARM                    FLAG
     C                   PARM                    RCODE
     C                   ENDSR
```

Fig. 1

Here is a fixed format example of code to pop information from the REXX Queue. The code retrieves the last entry from the stack and returns the data to the program via the buffer (BUF) parameter.

```
      *=================================================================
      * @POP - GET STACK DATA
      *=================================================================
     C     @POP          BEGSR
     C                   EVAL      FLAG = 0
     C                   EVAL      RCODE = 0
     C                   MOVE      'P'           FUNT                               Pull
     C                   EVAL      BUFLEN = 264
     C                   CALL      'QREXQ'
     C                   PARM                    FUNT
     C                   PARM                    BUF
     C                   PARM                    BUFLEN
     C                   PARM                    FLAG
     C                   PARM                    RCODE
     C                   MOVEL     BUF           COMAND
     C                   ENDSR
```

Fig. 2

In free-format code the call to the system API, QREXQ may be prototyped. (Fig.3)

```
      *-----------------------------------------------------------------
      * Prototype for QREXQ API.
      *-----------------------------------------------------------------
     D useRexQue       PR                  extPgm('QREXQ')
     D  rxQfunct                      1
     D  rxBfrVal                    128
     D  rcBfrLen                      9B 0
     D  rxBfrFlg                      4B 0
     D  rxRtnCde                      4B 0
      *-----------------------------------------------------------------
      * REXX queue DS for QREXQ API.
      *-----------------------------------------------------------------
     D QBUFFER         DS
     D  rxQFunct                      1
     D  rxBfrVal                    128
     D  rxBfrLen                      9B 0
     D  rxBfrFlg                      4B 0
     D  rxRtnCde                      4B 0
```

Fig. 3

However, the process is the same as in fixed-format. Send an entry to the queue, from the buffer when pushing an entry onto the queue and pull the entry from the queue when retrieving the information.

```
          DoU rxRtnCde <> 0              ;
             rxBfrLen   = 128            ;
             rxQFunct   = 'P'            ; // Pull an entry from the queue
             rxRtnCde   = 0              ;
             rxBfrFlg   = 0              ;
             useRexQue(rxQFunct          :
                    rxBfrVal             :
                    rxBfrLen             :
                    rxBfrFlg             :
                    rxRtnCde            );
             IF rxRtnCde = 0             ;
                rowCount = rowCount + 1  ;
                %occur(fmtlin) = rowCount; // count of late pulls
                fmtLin = rxBfrVal        ;
             ENDIF                       ;
          ENDDO                          ;
```

Fig.4

The code in the example (Fig. 4) demonstrates pulling information from the REXX queue. The loop will read all of the records from the queue, emptying the data queue. Though the queue itself cannot be deleted by a program, the contents may be emptied using a logic loop such as the one above.

In order to push data to the queue, a loop such as the example below, (Fig. 5) may be coded. This code example is checking information from a multiple occurrence data structure and pushing the information to the REXX queue.

```
            DoU x > rowCount                     ;
               IF x <= rowCount                  ;
                  %occur(fmtLin) = x             ;
                  rxQfunct = 'A'                 ;
                  rxBfrlen = 128                 ;
                  rxBfrFlg = 1                   ;
                  rxRtnCde = 0                   ;
                  rxBfrVal = fmtLin              ;
                  useRexQue(rxQfunct: rxBfrVal:
                     rxBfrlen: rxBfrFlg: rxRtnCde);
               ENDif                             ;
               x = x + 1                         ;
            ENDDo                                ;
```

Fig. 5

The REXX queue is a flexible, fast mechanism for storing temporary data, or passing data from program-to-program within a job stream without resorting to a physical file or parameters. The caveat to remember is that the REXX queue is a job-related queue. When the job ends, so does the existence of the queue and any data that it contains.

- SC