# Burroughs

# PUBLICATION CHANGE NOTICE

PCN No.: 1090685-003 Date: November 1982

Publication Title: B 2000/B 3000/B 4000 Series System Functional Description Manual (April 1982)
(Relative to ASR 6.6 Release Level)

Other Affected Publications: None

Supersedes: N/A

Description: This PCN makes changes to the basic publication. In addition, the title of the basic publication is changed by this PCN (formerly B 2000/B 3000/B 4000 Series MCPVI System Software Programmer's Guide).

Revisions to the text are indicated by a vertical black bar on the affected pages..

| Replace these pages | Add these pages |
|---|---|
| Title | xvii |
| iii | 8-19 thru 8-21 |
| ix thru xv | 9-10A thru 9-10D |
| 2-1 | 10-10A thru 10-10B |
| 2-9 | 11-1 thru 11-15 |
| 2-27 | |
| 2-83 | |
| 3-5 | |
| 3-15 thru 3-17 | |
| 4-29 | |
| 5-5 | |
| 5-21 | |
| 5-29 | |
| 5-41 | |
| 8-1 thru 8-17 | |
| 9-5 thru 9-11 | |
| 10-1 thru 10-9 | |

# LIST OF EFFECTIVE PAGES

| Page | Issue | Page | Issue |
|---|---|---|---|
| Title | PCN-003 | 5-7 thru 5-20 | Original |
| ii | PCN-003 | 5-21 thru 5-22 | PCN-003 |
| iii thru iv | PCN-003 | 5-23 thru 5-28 | Original |
| v thru viii | Original | 5-29 thru 5-30 | PCN-003 |
| ix thru xv | PCN-003 | 5-31 thru 5-40 | Original |
| xvi | Blank | 5-41 thru 5-42 | PCN-003 |
| xvii | PCN-003 | 5-43 thru 5-69 | Original |
| xviii | Blank | 5-70 | Blank |
| 1-1 thru 1-2 | Original | 6-1 thru 6-7 | Original |
| 2-1 thru 2-2 | PCN-003 | 6-8 | Blank |
| 2-3 thru 2-8 | Original | 7-1 thru 7-10 | Original |
| 2-9 thru 2-10 | PCN-003 | 8-1 thru 8-22 | PCN-003 |
| 2-11 thru 2-26 | Original | 9-1 thru 9-4 | Original |
| 2-27 thru 2-28 | PCN-003 | 9-5 thru 9-10 | PCN-003 |
| 2-29 thru 2-82 | Original | 9-10A thru 9-10D | PCN-003 |
| 2-83 thru 2-84 | PCN-003 | 9-11 thru 9-12 | PCN-003 |
| 2-85 thru 2-94 | Original | 9-13 thru 9-34 | Original |
| 3-1 thru 3-4 | Original | 10-1 thru 10-10 | PCN-003 |
| 3-5 thru 3-6 | PCN-003 | 10-10A thru 10-10B | PCN-003 |
| 3-7 thru 3-14 | Original | 10-11 thru 10-24 | Original |
| 3-15 thru 3-18 | PCN-003 | 11-1 thru 11-16 | PCN-003 |
| 3-19 thru 3-21 | Original | A-1 thru A-7 | Original |
| 3-22 | Blank | A-8 | Blank |
| 4-1 thru 4-28 | Original | B-1 thru B-2 | Original |
| 4-29 thru 4-30 | PCN-003 | C-1 thru C-3 | Original |
| 5-1 thru 5-4 | Original | C-4 | Blank |
| 5-5 thru 5-6 | PCN-003 | | |

Burroughs

# B 2000/B 3000/ B 4000 Series

## System Functional Description Manual

**(RELATIVE TO ASR 6.6 RELEASE)**



PRICED ITEM

Printed in U.S. America

Cover Revised
November, 1982

1090685

The names, places and/or events depicted herein are not intended to correspond to any individual, group or association existing, living or otherwise and are purely fictional. Any similarity or likeness of the names, places and/or events with the names of any individual, group or association existing or otherwise is purely coincidental and unintentional.

You should be very careful to ensure that the use of this software material and/or information complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this publication should be forwarded using the Remarks form at the back of the manual, or may be addressed directly to TIO West Documentation, Burroughs Corporation, 1300 John Reed Court, City of Industry, California 91745, U.S.A.

# TABLE OF CONTENTS (Cont)

# TABLE OF CONTENTS

# TABLE OF CONTENTS (Cont)

# TABLE OF CONTENTS (Cont)

# LIST OF ILLUSTRATIONS

# LIST OF TABLES

# LIST OF TABLES

# INTRODUCTION

This manual describes the various programmatic interface areas available with MCPVI. The intent of the information is not to detail the operations of the MCP, but to consolidate the information which a programmer might need, for example, in reading a dump, or in writing a user-coded utility program.

The user of this manual presumably has some programming knowledge and operating experience with B 4000/B 3000/B 2000 series MCP. Because of the constructs used in many explanations, a familiarity with Medium Systems Assembler will be useful.

The reader is also referred to the MCPVI Software Operations Guide, Volume 1 form number 1127529 for specific syntax and semantics of keyboard and control messages used in this manual.

The information contained in this manual replaces the information contained in the following Medium Systems Technical Newsletters: form numbers 1042272-010, −013, −024, −025, −027, and −028. The information contained in these technical newsletters still applies to MCPV.

# SECTION 2
# PROGRAM INTERFACE

A user program communicates with the MCP using a Branch Communicate instruction (BCT, OP = 30). With a BCT, a program requests the MCP to perform one of a variety of functions such as file OPEN, sort calls, or physical I/O. This section describes the user program Branch Communicates.

Also included are the interface requirements for various system utilities that can be user-written.

## PROGRAM BRANCH COMMUNICATE

The BCT instruction is used to transfer control from a user program to the MCP. The address specified in the BCT is an absolute memory address within the MCP. A BCT address is valid if the first digit at that address is an undigit @F@. If the first digit is not @F@ a processor interrupt occurs. The five digits following the @F@ is an address in the MCP which will handle that particular BCT. The remaining five digits of the address must also be valid.

Each Communicate is headed by the equivalent COBOLV/ COBOLL construct (where such exists). Following in parentheses, is the BPL term for the communicate.

The Communicate structure is described as follows:

```
BCT   Address
BUN
P1
  •
  •
  •
Pn
```

The BUN does not imply that the instruction must be an unconditional branch, but rather, a branch format (8-digit or 10-digit) instruction. However, an unconditional branch is generated by the compilers for all communicates where BUN is indicated.

All parameters are labelled P1 through Pn, including fillers. ACONs in the parameters must not have any address controller, extended address, or indexing specifications, and cannot exceed the program limit register. Addresses of File Information Blocks (FIBs) cannot exceed 399998. ( In some cases, ACONs in programs compiled prior to MCPV, contain UA address controller specifications. Earlier MCPs ignored the high-order digit in certain communicates. As this digit is significant for MCPVI, if the 6-digit address is found to exceed the limit of the program, the MCP ignores the 2-bit of the high-order digit.) In some cases ACON parameters are optional, signified by a zero value in the field. End-of-file (EOF) labels are an example. (However, in the case of EOF addresses, one must be present where needed, namely when EOF is reached.)

## ACCEPT (ACCEPT)

The ACCEPT BCT requests data from the Operator Control Station (OCS).

The format of this BCT is:

```
      BCT     0254
      BUN
P1.   ACON    (input area, must be mod 2)
P2.   CNST 1  UN = 0 (flag indicating ACCEPT)
P3.   CNST 2  UN = XX (size of input area in bytes)
P4.   CNST 1  UN = 0 Filler
```

P1 must be MOD 2. P1 cannot exceed 60 bytes. If P3 is equal to zero or greater than 60, 60 is used.

The AX OCS command may be entered and saved prior to a program executing the ACCEPT BCT. If input text from a previously entered AX command is available, the text is inserted into the input area and the program is reinstated immediately. If the size of the text exceeds P3, the text is truncated and no error message is displayed.

If no previous text is available, the program is suspended until an AX command is entered. If the number of characters entered exceeds P3, the operator is notified with an error message on the OCS and ACCEPT is repeated.

In either case, if fewer than P3 characters are input, the data is left-justified in the buffer. An ETX (03) follows the last character input. The buffer contents following the ETX are undefined. If exactly P3 characters are input, no characters are inserted.

## CLOSE (CLOSE)

The CLOSE BCT closes files and designates their disposition. The format of this BCT is:

```
      BCT     0154
      BUN
P1.   ACON    (FIB)
P2.   CNST 1  UN  =  X (type of CLOSE-Parameter 1)
                0 =  FILE (REWIND)
                1 =  REEL (REWIND)
                2 =  FILE NO REWIND
                3 =  REEL NO REWIND
                4 =  FILE RELEASE
                5 =  REEL RELEASE
                6 =  FILE LOCK
                7 =  REEL LOCK
                8 =  FILE PURGE
                9 =  REEL PURGE
                C =  FILE CLOBBER
                E =  FILE CLOBBER
P3.   CNST 1  UN  =  Y (type of CLOSE-Parameter 2)
                1 =  CRUNCH P2 must be 4, 6, C, or E)
                2 =  NO REWIND RELEASE  (P2 must be 4)
                4 =  COBOL-74 FILE LOCK
                8 =  NO DISCONNECT
```

CLOSE REEL and NO REWIND operations are defined for magnetic tape and paper tape only.

CLOSE CLOBBER and CRUNCH are defined for output disk files or disk pack only. CRUNCH can be ignored if more than one area of the file is allocated.

NO DISCONNECT applies only to dialed data communications devices.

CLOSE PURGE applies only to disk and disk pack files, and output magnetic tape files.

COBOL -74 FILE LOCK applies only to COBOL-74 programs. If a program reopens a file that has been CLOSED with LOCK, it will be terminated with an INVLOCK USE message.

## COMPLEX WAIT (NO COBOL OR BPL SYNTAX)

The COMPLEX WAIT BCT allows a user to suspend execution until one of the specified events happens. If any event has happened at the time the BCT is executed, the program is reinstated immediately.

The COMPLEX WAIT BCT is not allowed to a program which is executing in the shared area.

The format of the BCT is:

```
      BCT        0994
      BUN
P1.   CNST 2UN   = XX (number of parms in list)
P2.   ACON       (address of parameter list)
P3.   ACON       (address of 2 UN using field or 0 if none)
P4.   ACON       (address of 2 UN giving field or 0 if none)
```

P2 points to the list of parameters which is defined as follows:

```
CNST 4UN   = XXXX (event type)
             0 = time (doze)
             1 = ODT input present
             2 = output event
             3 = input event
             4 = change event
             5 = ready event
ACON       (depends on event type)
ACON       (depends on event type)
```

If event type = 0, the first ACON points to a 5 UN time field in seconds and the second ACON is zero. If event type = 1, both ACONs are zero. If event type = 2, 3, 4, or 5, then the first ACON points to a FIB, and if the file is a port file, then the second ACON points to a 4 UN subport index. If the subport index is zero, this is a port event.

NOTE
An output event is invalid as a port event.

These three fields are repeated P1 number of times and must be contiguous.

When the BCT is executed, each event is tested to see whether it has already occurred. The order in which the events are tested is determined by P3. If P3 = 0, the events are tested in the order they are specified in the list. Otherwise, the first event tested is the one that occupies the P3-rd (1 relative) position in the list.

When the program is reinstated after executing a COMPLEX WAIT BCT, it has the option of requesting an indication of which event has happened. If P4 = 0 no indication is given; otherwise the (1-relative) position in the original list of the event that occurred is placed in the giving field.

When a program executes a COMPLEX WAIT BCT the MCP builds an entry in the Complex Wait Table for that program (refer to section 5). If there are not enough entries available in the Complex Wait table, the program is suspended while waiting for Wait Table space. However, if Wait Table space does not become available, the BCT is invalid and the program is terminated. The MCP's 100-second status routine allows any program waiting for Wait Table space to re-execute the COMPLEX WAIT BCT if space becomes available. The operator may cause an earlier re-execution of the BCT by issuing a <mix>OK command from the OCS.

## MIXTBL (NO COBOL SYNTAX)

The MIXTBL BCT requests information about current jobs in the mix. Programs running in the time-share area that execute this BCT will only receive information about jobs in the time-sharing mix. The format of this BCT is:

```
     BCT          0214
     BUN
P1.  ACON         (program ID field, table, or unused depending on P3)
P2.  CNST 1 UN    = 6 (specifies PROGRAM request)
P3.  CNST 1 UN    = X (defines type of request)
P4.  CNST 2 UN    = YY (holds response)
```

The values of P1 and P3 can be:

| P3 | P1 | Request |
|---|---|---|
| 0 | Program ID Address | Place number of programs in mix with specified ID in P4. |
| 1 | Filler | Place number of programs in mix in P4. |
| 2 | Program ID Address | Place <mix-no.> of specified program in P4. |
| 4 | Table Address | Place mix information in specified table; also put number of programs in mix in P4. |
| 8 | Filler | Place <mix-no.> of caller in P4. |
| 9 | Response Area | Place caller information in P4. |

Following is the table format (P3 = 4).

**Header**

| | |
|---|---|
| Reserved | 1 UN |
| Jobs in Mix | 2 UN (value also in P4) |
| Core Available | 3 UN (mod 1000; total available memory) |

**Body (one entry for each program)**

| | |
|---|---|
| MIX ID | 6 UA (program name) |
| MIX-MF | 6 UA (multi-program name) |
| MIX-NO | 2 UN (mix number) |
| | 1 UN (reserved) |
| | 3 UN (memory used by job, base to limit only) |

The program ID field (where relevant) must be 6 UA, mod 2, and must contain the name of the program in question.

It is the responsibility of the programmer to allocate a table of sufficient size when P3 = 4.

For P3 = 9 the response area contains the following information:

| | |
|---|---|
| 6 UA | Program ID |
| 6 UA | Multi-program ID |
| 2 UN | Mix number |
| 4 UN | RLOG number |
| 4 UN | RJE link |
| 6 UN | Date compiled (mmddyy) |
| 5 UN | Time compoled (hh:mm) |

If the requestor has been inititated through RJE, the RJE link should be used when creating pseudo-decks.

Any program that creates pseudo-decks should execute this BCT with P3 = 9 to obtain an RJE link. If the link is non-zero, the program should place an @FF@ in column 74 and the link in columns 75-76 of the control cards in the pseudo-decks it builds.

# OTHER USER PROGRAM INTERFACES

In addition to the specific information passed to the MCP in a BCT instruction, other interface areas are involved during the processing of a BCT. Two particular functions, USE routines and disk/disk pack file OPENs, modify or examine certain program locations. These additional parameters are presented here.

## USE ROUTINES

When certain USE routines are entered, the MCP passes information to the program which helps define the processing state at the time of entry. In all cases, the address to which return is made when the USE routine is exited, is placed in FIBRCW of the first Buffer Status Block following the FIB for which the USE routine was entered.

The following data is passed when a label USE routine is entered:

BASE:+34:1:UN = FIB-IO (value)
BASE:+35:1:UN = 0 for begin label, 1 for end label
BASE:+36:1:UN = 0 for file labels, 1 for reel labels

The following data is passed when an I/O error USE routine is entered:

BASE:+34:1:UN = FIBRWT bits :1 and :2
BASE:+37:3:UN = Reel number (MTP)

## FILE HANDLING

When an existing disk or disk pack file is OPENed input or I/O, the program can request that the MCP present the disk file header (DFH) for the file in memory. This is done by setting certain bit combinations in the first digit of the FIB for that file (FIBST1). The specifications are:

| FIBST1 Value | Meaning |
|---|---|
| :1 | Request first 40 digits of header before modification (due to remapping according to record and block size specifications) at BASE:+100. |
| :4 | Request first 40 digits of header after modification at BASE:+100. |
| :8 | Request header addresses at BASE:+140; the number of addresses transferred is determined by the number of areas specification in the DFH. Bit 1 or 4 will also be on. |

When a new output disk or disk pack file is CLOSEd WITH LOCK, RELEASE, or CLOBBER, the program can specify that the MCP is to access memory (at BASE:+100) for the first 40 digits of the DFH, rather than use the header maintained outside program bounds. This is specified by the 1-bit in FIBST1.

## MCP-UTILITY INTERFACES

The MCP provides several utility programs as intrinsics. Most of these programs can be user written and employ the standard MCP interface for that intrinsic. It must be understood that the interfaces for the intrinsics defined in the following are subject to change. This sub-section is intended primarily for information.

### Print/Punch Backup

A complete description of backup utilities interfaces can be found in Device Alternates, section 4.

### Log Programs

The names of the user-coded log analysis programs (initiated by the LNR, LNS, and LNM messages) must be RLGOUT, SLGOUT, and MLGOUT, respectively. The 2-digit log number (p1-p9) is inserted at BASE:+32 (p = processor number).

### Disk File LOAD/DUMP

The user-coded LOADMP is invoked in lieu of the MCP intrinsic except when the file name specified in the control syntax is SYSTEM (and /OWN is not specified), or when /MCP is specified.

The parameters passed to the program when it is initiated are shown in table 2-4.

**Table 2-4. Disk File LOAD/DUMP Parameters**

| Location | Length | Meaning |
|---|---|---|
| BASE:+0 | 1UN | Execution digit<br>0 = LOAD<br>1 = DUMP<br>2 = ADD<br>3 = UNLOAD<br>4 = CHECK<br>5 = DUMP/CHECK<br>7 = UNLOAD/CHECK<br>:8 = LOAD for ALLOCATE card (that is, OPEN files I/O) |
| BASE:+1 | 1 UN | :2 NEWLIST and LIB = 1, or LIST specified in control request<br>:4 Abort on error<br>:8 Tape FID = SYSTEM |
| BASE:+2 | 2 UN | Segments per block for 7-track (1-30) |
| BASE:+2 | 1 UN | High order bits of size field<br>:4 DISKCHECK on LOAD/ADD<br>:8 COMPARE on DUMP/UNLOAD |
| BASE:+4 | 2 UN | Specified EU for LOAD/ADD |

In addition to the parameters passed to program memory, the MCP builds a work disk file for LOADMP. This file, which consists of 100-byte records (20 X 40), contains FIDs and other data needed for the requested operation. The file name is %nn0p0, where nn is the LOADMP mix number and p is the system processor number. The record format is shown in table 2-5.

The FILL...INTO at WAKE-UP is used strictly as a means to interlock with BPLSCN and to alert the program that a new record is to be obtained. It is possible to have multiple programs using BPLSCN simultaneously (for example, by making the BPLSCN FILL's global requests or by passing the program name with the COBOL FILL INTO).

Observe that asynchronous communications yields considerable benefits for the above example. The following sub-section examines this example further.

## ASYNCHRONOUS PROCESSES

Both CRCR and STOQUE provide tools for efficient asynchronous process control; each mechanism has applicational advantages. The two mechanisms can be distinguished by the fact that CRCR provides synchronized data transfer while STOQUE permits completely asynchronous exchanges. Programs using CRCR can process any given transaction asynchronously but must be synchronized at the time of data transfer. The latter is not necessary when using STOQUE. In addition, each mechanism has a number of other unique characteristics.

When CRCR is used for asynchronous processing, the PROCEED TO... clause of the FILL...INTO and FILL...FROM constructs are often employed. This facility allows a process to pursue other activities while the other program completes processing preparatory to the hook up. This capability is important in applications requiring a handler program. Such a program can be servicing many users (peripheral devices and/or programs) and cannot wait for a hook up with one program while other users require servicing. At a later time the CRCR request can be repeated; this process of executing the PROCEED TO... clause and later reinitiating the request is called buzzing. Since CRCR requires that the data transfer be synchronized, only one of the processes involved can specify the PROCEED TO... clause for a given transfer; otherwise a hook up would never occur. Thus, at times, one process must be waiting for the other to catch up.

The CRCR requests have another function which is sometimes valuable in asynchronous processing, usually in conjunction with the normal data transfer facility. If a program attempts to FILL...INTO or FILL...FROM another program and no process of the specified name is prepared to receive or send data, but a program of that name is either a WAIT or SLEEP status, the WAITing/SLEEPing program is marked ready-to-run. Thus, non-Global CRCR operations can be used to wake up another program. (A WAIT [UNTIL <time>] or WAIT <time> request, respectively.)

The STOQUE functions of the MCP are performed by the STOQUE extension module of the MCP. If these facilities are used, they must be loaded into memory by setting the STOQ system option.

The STOQUE module performs the basic functions of transferring data from a data area in a program to an external memory buffer and retrieving that data upon request. The mechanism can be used simultaneously by any number of programs as a means to transfer data between processes, or even as temporary storage for a single process. The data elements placed into the memory buffer are organized into one or more program-independent, symbolically-named lists called storage queues (STOQUE).

The STOQUE mechanism differs significantly from CRCR in that no synchronization of the sending and receiving programs is required. This is due to the fact that STOQUE does not transfer data directly from one program to another, but instead stores the message in an external memory area until requested. Therefore, multiple transactions can be in the storage queue simultaneously, thus, the use of STOQUE permits the complete overlap of processing between programs, with no necessity to interlock for each transaction.

Programs can execute three types of calls on the STOQUE module: store data, retrieve data, and queue inquiry. Both the storage and retrieval functions have important variations which permit the mechanism

to be used in many ways. Rather than permitting only simple queuing of data, STOQUE allows elements to be added to or removed from a list at either the top or bottom yielding the benefits of both first-in first-out (FIFO) and last-in first-out (LIFO) mechanisms. In addition, queues can optionally be given a substructure to allow priority handling of data elements by associating a name with the individual elements of a list in addition to the individual name of the queue.

The queue inquiry function provides a rapid means of determining the size of a list without disturbing any elements.

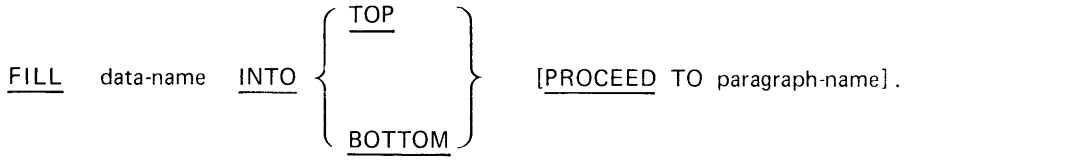
All requests made of the STOQUE function, reference an area in the user program called the STOQUE Parameter Block. This program-maintained area contains the information needed by STOQUE to control the data elements, and is described as follows:

```
QUEUE NAME           PIC X(6).          Identifies individual queue
ENTRY NAME LENGTH    PIC 9(2) COMP.     Name LGH in bytes (00=NULL)
ENTRY NAME           PIC X(NN).         Entry name (optional)
ENTRY DATA LENGTH    PIC 9(4) COMP.     Data LGH in bytes
ENTRY DATA           PIC X(NNNN).       Data field (optional)
```

The queue name identifies the programmatically assigned symbolic name of the list to which the request pertains. The entry name length field specifies the size of the optional entry name field. If present, the entry name specifies the name associated with the individual queue entry. This name can be used to provide a substructure to a list and provides the means to access data elements which are at other than the top or bottom of the queue. The entry data length field specifies the size of the entry data area which contains the transaction to be accessed for a storage request or is the area to which data is placed in a retrieval operation. The entry data length field functions as a response area for a queue inquiry request. The entry data field, however, is not applicable to a queue inquiry request. The Stoque Parameter Block must begin on a mod-4 address.

The following paragraphs provide a brief description of the STOQUE requests as illustrated by the COBOL language syntax.

To store data in a STOQUE, the COBOL syntax is:




The keyword INTO specifies that the request is for data storage. If the data element is to be queued at the head of the queue, TOP is specified; BOTTOM indicates that the entry is to be queued at the base of the list.

The data-name references the STOQUE Parameter Block described above. This block contains the data to be stored, and information needed by STOQUE to identify the queue and control the entry.

When the request is completed, the program is resumed at the next instruction.

If sufficient space is not available for the storage request, the sending program is suspended until space becomes available, unless the optional PROCEED TO clause is specified. In that case, the program does not WAIT, but resumes at the paragraph-name specified.

programs WAITing CRCR on this program are in receive mode. For that reason, the WAKE-UP mechanism is generally used in conjunction with the other functions of WAIT or SLEEP, or when proper timing can be assured. Note that the WAKE-UP occurs only on non-Global SEND or RECEIVE requests; the PROCEED TO option can be used or not, as desired.

A program must never buzz without giving up control to another process. If a program simply loops on one or more FILL requests, it is always reinstatable; further, as the highest priority job on the system, no other program, including the one for which the buzzing job is waiting, can ever execute. Even if it is not the highest priority job, the program would expend less overhead if other tasks could process between the FILL requests.

## STORAGE QUEUE

The following variables must be considered when using STOQUE.

## STOQUE Parameter Block

When a program executes a STOQUE request, several parameters must be supplied: the request type, the name of the queue to which the request pertains, the entry name (if used), and the data area to be used in conjunction with the storage or retrieval request. The request type parameter is part of the STOQUE statement. All other elements are constructed by the program in a STOQUE Parameter Block which is referenced in the STOQUE request. The block must begin at a mod-4 address (for example, a COBOL 01 level). The first six bytes contain the name of the queue (Stoqname) to which the request pertains; this name must be left justified and blank filled if less than six bytes long. The name can contain any EBCDIC characters, including embedded blanks and non-graphic combinations, except that the entire name cannot consist of all NULLs or be MCPQUE (reserved for MCP). The next element is a 2-digit field containing the size of the entry name in bytes. The field can contain any value from 0 to 99; if no entry name is used, the length should be zero. The entry name field then follows. The name, which can consist of any EDCDIC characters, can be used to provide a substructure to a queue. Multiple entries in a queue can have the same entry name.

Retrieval requests need not specify the complete name associated with an entry. If a shorter name is used, the name acts as a group specifier as described earlier.

The length of the message area is specified in a 4-digit field containing the size of the buffer in bytes. The length can be any value from 1 to 2300. For a POLL operation, this field is used as the response area rather than a data length field as no message area exists for this operation.

The message area follows the length field. No restrictions are placed on the content of the field. A maximum length of 2300 bytes is permitted.

When a storage request is made, all fields from the entry name length field to the end of the entry data field are moved to the queue. When a retrieval request is made, these fields are moved from the queue entry to the program STOQUE Parameter Block Area. In certain cases, care must be exercised in retrieval request using group entry names. For example:

```
01  STOQ-PAR-BLOCK.
    03 QNAME PIC X(6) VALUE "QNAME1"
    03 QENT-NAME-LENGTH PIC 99 COMP VALUE 03.
    03 QENT-NAME PIC XXX VALUE "ABC".
    03 QENT-DATA-LENGTH PIC 9(4) COMP VALUE 0100.
    03 QENT-DATA PIC X(100).
```

If an entry in QNAME1 has an entry name of ABCD, a retrieval request for ABC can cause an unexpectedly large entry to be accessed. For this case, QENT-NAME-LENGTH would contain 04, QENT-NAME would be a 4-byte field containing ABCD (spilling over into QENT-DATA-LENGTH) and all other fields would be shifted over one byte.

Proper access of the data can be accomplished with a redefinition of STOQ-PAR-BLK. Retrieval requests using an entry name are slightly less efficient than simple requests. Storage requests are equally efficient, regardless of type.

## Buffer

The memory buffer maintained by STOQUE for the storage of data is allocated in user memory external to all programs. The space is allocated as needed and deallocated when possible. The user can control the amount of memory which STOQUE can use for storage space through Cold Start LIMIT specifications.

Memory is obtained in variable size blocks (5 to 20KD each) called Stoqblocks. The number of these Stoqblocks which can be allocated, up to a maximum of 980, is controlled by two LIMIT specifications (STOQSIZE and STOQBLOCKS). If it is not possible to get enough memory space for a Stoqblock, the message ** NO MEM FOR STOQ is displayed on the OCS. To minimize the overhead of frequent memory allocation and deallocation, the MCP checks for empty Stoqblocks on a periodic basis rather than returning them when empty.

## Queue Names

Data elements placed into the memory buffer by the STOQUE module carry no information about the program from which it was sent or destined. Instead, the elements are organized into a queue identified by the progammatically specified Stoqname. The Stoqnames are maintained in a table contiguous to the STOQUE module. Because the Stoqname defines a queue, the capacity of the name table limits the number of separate queues which can be maintained simultaneously. The STOQUE module provides a table with a capacity of 10 names by default; a LIMIT STOQNAMES parameter card can be used to increase the table size beyond the default value up to a maximum of 99 entries. Because memory for any additional table space is allocated in 1 KD increments, maximum efficiency of memory utilization occurs when the Stoqnames table is 10, 43, or 99 entries in size as each entry is 30 digits long.

A new entry is made in the Stoqnames table when the first storage or retrieval request is made for an entry of that name. An entry is not deallocated until the name table is full, a program executes a request for a new queue name and a queue is completely inactive (no entries exist and no programs are waiting for entries), or until an RQ keyboard message is entered.

## Queue Organization

The queue entries can occur anywhere in the STOQUE buffer space. Inter-element links (addresses) provide the structure and organization of a queue and thus allow a simple means to access elements or place new items into a queue. This type of structure is known as a linked list.

The organization of the storage queues is shown graphically in figure 3-1. The entries of the Stoqnames table provide rapid access to the list at either the head or base. In addition, each list element points to both the next and previous entry, allowing access to any individual item in the queue through either the head or base. The list, then, is bi-directional.

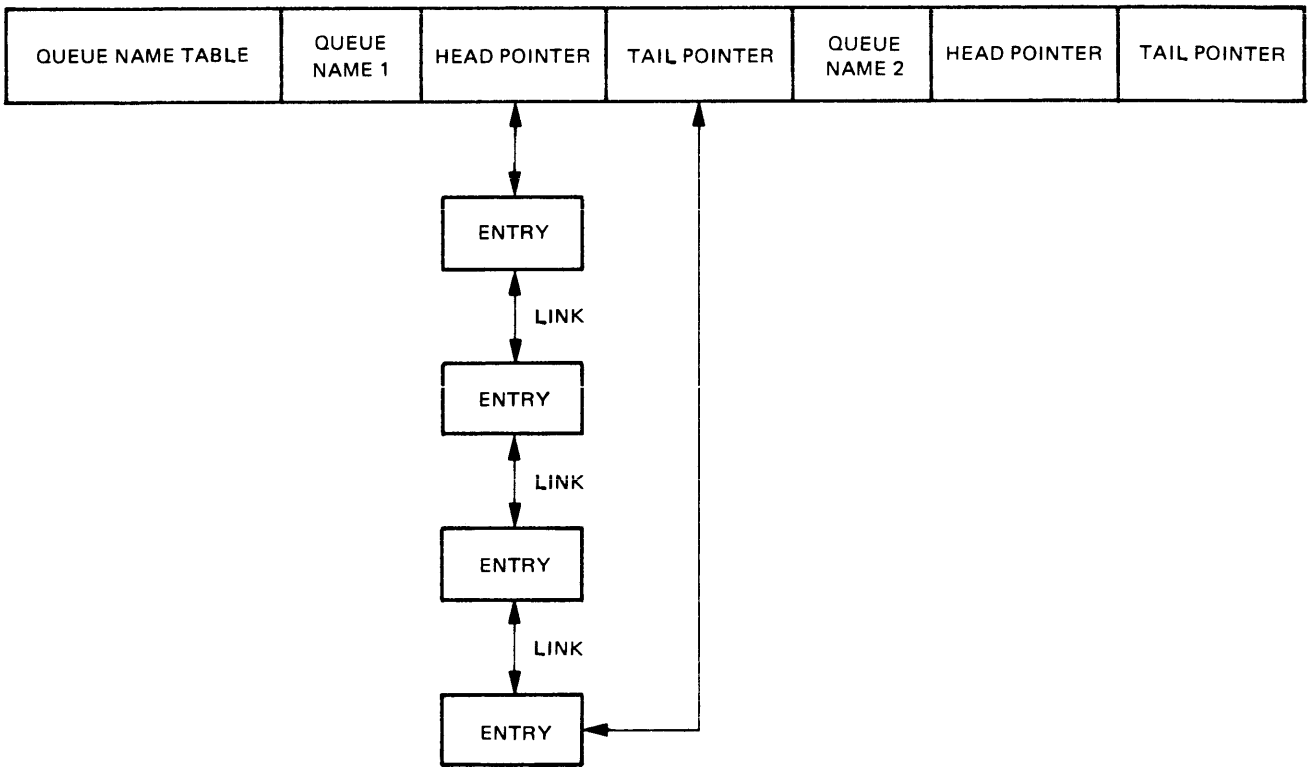



**Figure 3-1. Storage Queue Organization**

The flexible structure of the queues permits a wide range of applications. The existence of both forward and backward links allows items to be added to or retrieved from either the head or base and thus permits a queue to be used as a sequential list (FIFO), or as a push-down stack (LIF0), or both. Extremely complex queuing and retrieval mechanisms can be developed by combining these operations. Even more sophisticated designs are possible by using queue entry names as well.

## Storage Queue Entries

When a program executes a storage request, STOQUE obtains the necessary buffer space and moves the program data to the entry. In addition, a number of control fields are established in the entry and are used to manage the empty space. STOQUE then links the entry into the proper queue in the manner specified by the program.

Similarly, a retrieval request causes data to be moved from an entry to the requestor. The entry is then returned to the available space list.

The queue entry contains several important components which are used for the following purposes:

Space management
: Links to the next and previous physical space in the Stoqblock and size specification are used when entries are retrieved and the space is to be returned to the available list.

Queue management
: Links to the next and previous entries in the queue are maintained to permit access to elements in either direction.

Entry management
: The size of the user portion of the entry and the entry is present.

Filler
: Filler may be present to make the entry mod 4 digits in size and/or because the original available slot was not large enough for both the entry and new available links for the remainder.

The user portion of the entry is simply the program specified STOQUE Parameter Block (except the queue name portion). The MCP is unconcerned with the contents of this area except at the time of a retrieval request with an entry name specification. The requested name is checked against a corresponding number of characters of the entry name in the queue entry until a match is found or the queue is exhausted.

The user portion of an entry can range from 4 to 2403 bytes in length depending on the size of the entry name and data area. Retrieval operations are more efficient if the size is an even number of bytes.

### Available Space List

In addition to the active queues, available areas within the buffer space are maintained in a linked list. When a storage request is made, the STOQUE module attempts to find the smallest available area which can contain the entry. If no available space is large enough, a new Stoqblock is obtained (subject to the user-specified block limit and memory availability) and linked into the available space list.

As entries are removed from a queue, the MCP returns the space to the available list, consolidating the entry with surrounding space as applicable.

If an entire block remains in the available list for 40 seconds, the memory space is returned to the system, unless this would reduce the number of remaining STOQUE blocks below the minimum specified on a Cold Start LIMIT card. This mechanism minimizes frequent memory allocation and deallocation operations.

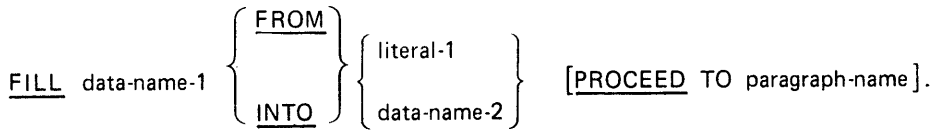
At any given moment, available space can be extensively checker-boarded throughout the Stoqblocks; because of the transient nature of the individual entries, the available list changes rapidly. Thus, no attempt is made to consolidate the available areas within a block.

## LANGUAGE CONSTRUCTS FOR CRCR AND STOQUE

The language constructs for CRCR and STOQUE are presented here.

### CORE-TO-CORE CONSTRUCTS

In COBOL ANSI-68, the Core-to-Core construct is



(stacked) tape can be printed (PB <cc/u>) or 2) a specific file can be printed (PB <cc/u> <file-identifier>).

The operator can include a /OWN or /MCP parameter to require the initiation of either the user-coded or MCP version of the utility. In the absence of this parameter, the user-coded version is initiated if present on disk; otherwise, the MCP intrinsic is scheduled.

When the operator initiates printing, restart or skipping can be selected. This is done by a variant of the PB message: PB <cc/u> * <file-identifier> where the asterisk indicates that the operator wishes to specify a restart point in the file, or a number of tape records are to be bypassed before printing begins. (PB <cc/u> * without a file-identifier indicates a restart for all files on the tape.) The next specification in the PB request can be a 1 or 2-digit integer (0-99) which is passed to the print program. This number is not currently used by the MCP supplied utility but can be employed in any manner desired in a user-coded PBTOUT (see below). Full syntax for the PB message is:

PB [/OWN / /MCP] <cc/u> [*] [<file-identifier>] [<integer>] [; parameter instructions]

The parameter instructions can include any valid program parameter statement such as CHARGE, LOCK, AFTER, and so on. If the Cold Start CHRG ALL specification has been made, the CHARGE parameter instruction must be included.

*OPERATION AND CONTROL*

When the print program begins execution, the tape on the appropriate channel and unit is OPENed. If the operator has requested the printing of a single file on a stacked printer backup tape, the MCP multifile search routine finds the appropriate file; otherwise, the first file on the tape is selected. The label record is accessed to determine if FORMs and/or labels are required for the printer file. If a restart (*) is specified, the print program requests the necessary data from the operator and positions the file to the desired point.

After necessary positioning of the file, a printer is OPENed and printing begins. During printing, the operator can use the SK and QT messages to initiate special actions in the print program as described above. (When printing a PBT, for the last (or only) file on the tape, QT and QT ALL mean the same thing; however, if QT is used, PBTOUT may have to space to the end of the current file to determine if any more files are present.) When printing of the files on the tape is completed, the tape is rewound, but not purged, to allow multiple printings.

## Backup Disk/Disk Pack (PBDOUT)

The following tells how PBDOUT can be used.

*INITIATION*

PBDOUT is initiated by a keyboard message (PB, PBD, or PBP...) The operator can indicate that all files are to be printed in sequence according to numeric designations (PB/ , PBD/ , or PBP/), or that a specific file is to be printed (PB , PBD , or PBP <integer> ) where the integer is the numeric portion-nnnn or pnnnn-of the PBD file name. The pnnnn format can be used to print a file on a processor other than its creator.

The operator can include a /OWN or /MCP parameter to require the initiation of either the user-coded or MCP version of the utility. In the absence of this parameter, the user-coded version is initiated if present on disk; otherwise, the MCP intrinsic is initiated.

Restart or skipping can be specified by typing PB*, PBD*, or PBP*... ; the asterisk indicates that the operator wishes to specify a restart point or a number of records to be bypassed before printing begins. PB*/, PBD*/, or PBP*/ indicates restart for all backup disk/pack files.

The operator can also specify that the PBD files are to remain on or off disk after printing by specifying the word SAVE if the files are not to be purged. If no specification is made, the files are not SAVEd.

The next specification in the PBD request can be a 1 or 2-digit integer (0-99) which is passed to the print program and indicates the total number of copies of the backup file to be printed. If either a zero or a one is entered, or if the parameter is omitted, PBDOUT generates one copy of the file. Full system syntax for the PBD message is

```
PB [P] [/OWN] [*] { / }                [SAVE] [ <integer> ]
   [D] [/MCP]     { <file-name> }
                  { <file-number> }
[ ; <parameter instructions> ]
```

If the Cold Start CHRG ALL specification has been made, the CHARGE parameter instruction must be included.

When the printer program is initiated, the MCP passes it the file number (zeros if the message was PB/, PBP/, or PBD/), flags if *, and/or SAVE were specified and the copy parameter. The MCP does not establish that the desired file is in the Disk Directory before initiating the print program.

*OPERATION AND CONTROL*

When the print program begins execution the appropriate disk file is OPENed. If a particular file is specified, it is accessed; otherwise, the PBD file with the lowest number for that processor is requested of the MCP.

The first record is read to determine if FORMs and/or labels are required for the printer file. If restart (*) is requested, the print program requests the necessary data from the operator and postions the file to the desired point.

During the printing the operator can use the SK and QT message to initiate special actions in the print program as described above.

When a file has been completely printed, it is CLOSEd WITH PURGE unless the operator has specified SAVE in the PBD or PBP message. (The file is not PURGEd if QT or QT ALL is specified since the file is not completely printed for these cases.) If the original request was for the printing of a single file, PBDOUT terminates.

If the operator requests that all files be printed, PBDOUT requests the MCP to OPEN the backup print file with the next higher number for that processor. The process continues as described above until all PBD or PBP files on the processor have been printed or the operator terminates the program. (any PBD files CLOSEd by programs during the processing of a PBD/ request are also accessed and printed in turn without operator intervention.)

## DISK PACK FILE HEADER (ON DISK PACK)

The following descriptions apply to the disk pack version of the Disk File Header.

| Label | Relative Location /Size | Content/Meaning |
|---|---|---|
| PF-CAD | 0,6 | Zeros |
| PF-SPT | 6,8 | Address of this sector |
| PF-SIZ | 14,4 | Header size in bytes |
| PF-TP1 | 18,1 | File type |
| | | :8 <<Available>> |
| | | :4 File name change in progress |
| | | :2 Incomplete file (partially removed) |
| | | :1 Split cylinder file |
| PF-TP2 | 19,1 | File type |
| | | :8 Assign by space available file |
| | | :4 Assign by area file |
| | | :2 Single pack |
| | | :2/ Multipack file |
| | | :1 Cylinder bound file |
| PF-TP3 | 20,1 | File type |
| | | :8 <<Available>> |
| | | :4 If APCR is set, RN/ is inhibited<br>If APBD is set, PBP/ is inhibited<br>If APCM is set, PCP/ is inhibited |
| | | :2 <<Available>> |
| | | :1 No squash file |
| | 21,3 | Not used |
| PF-BEN | 24,8 | Block EOF pointer |
| PF-RSZ | 32,6 | Record size in digits |
| PF-RPB | 38,3 | Records per block |
| PF-BSZ | 41,9 | Block size in digits |
| PF-BPA | 50,6 | Blocks per area |
| PF-SPA | 56,6 | Sectors per area |
| PF-#AR | 62,4 | Areas requested |
| PF-UAR | 66,4 | Area counter |
| PF-EOF | 70,10 | EOF pointer |
| | | |

| Label | Relative Location /Size | Content/Meaning |
|---|---|---|
| PF-FRM | 80,2 | Record format |
| PF-USH | 82,8 | User header link |
| PF-PPA | 90,3 | Partitions per area (split cylinder files) |

Material Deleted by PCN 1090685-001

| | | |
|---|---|---|
| PF-CDT | 94,5 | Creation date |
| PF-LAD | 99,5 | Last access date |
| PF-SAV | 104,5 | Save factor |
| PF-ADB | 109,6 | Base pack header address |
| | 115,11 | Not used |
| PF-SNS | 126,1 | Sensitivedata flag |
| PF-STY | 127,1 | Security type |
| | | :8 Reserved<br>:4 Guarded<br>:2 Public<br>:1 Private<br>0 None |
| PF-SUS | 128,1 | Security use |
| | | 6 IO (default)<br>4 IN<br>2 OUT<br>1 SECURED |
| | 129,1 | Reserved |
| PF-SUC | 130,20 | Usercode |
| PF-GRD | 150,12 | Guard file ID |
| PF-OTY | 162,1 | Open type |
| PF-PRM | 163,1 | Permanent flag |
| PF-NU1 | 164,2 | Number of users processor 0 |
| PF-OO1 | 166,2 | Number of open out processor 0 |
| PF-NU2 | 168,2 | Number of users processor 1 |
| PF-OO2 | 170,2 | Number of open out processor 1 |
| PF-NU3 | 172,2 | Number of users processor 2 |
| PF-OO3 | 174,2 | Number of open out processor 2 |
| | | |

| Label | Relative Location /Size | Content/Meaning |
|---|---|---|
| | | Used as work area to build the absolute disk address for the current operation.<br>SEQ DSK:<br>Used as a work area to build the absolute disk address during disk positioning. |
| FIBORG | 110,1 | File Type |
| | | DSK:<br>:2 Indexed I/O key file.<br>:4 Indexed I/O data file.<br>:8 Relative I/O data file. |
| | 111,1 | Reserved |
| FIBDKB | 112,8 | Relative Block Number (redefines FIBDCF) |
| | | RND:<br>Temporary internal storage for relative block number of file during READ/WRITE requests. |
| FIBKEY | 120,6 | Actual Key Location (redefines FIBABS, FIBJAM, FIBMTL) |
| | | RND:<br>Address of actual key (8 UN).<br>MTP:<br>Used internally to store first six digits of I/O descriptor during tape position. |
| FIBSBL | 126,1 | Buffer Flag |
| | | :1 New area OPENed by GETDSK.<br>:2 DPK:<br>File is DPK file.<br>:4 HPT:<br>File is HPT or backup disk.<br>:8 All files except RND, DCM, SOR:<br>Buffer required flag; buffer exhausted on previous I/O request, new buffer needed for further processing (used for all work area access files except variable length). |
| FIBUNF | 127,1 | Random Disk Wait Flag; Tape Position Flag |
| | | RND:<br>Specifies program waiting for RND processing. |

| Label | Relative Location /Size | Content/Meaning |
|---|---|---|
| | | 1 = Buffer required for current request (I/O not yet initiated).<br>2 = Program waiting I/O on current request (READ or implicit SEEK for blocked WRITE).<br>3 = Program waiting buffer availability for next I/O (WRITE in process on buffer access file).<br>MTP:<br>Specifies program waiting for MTP position.<br>2 = Waiting for all in process I/Os to complete before positioning begins.<br>4 = Waiting for I/O on SPACE operation when moving tape.<br>6 = Input file buffer fill in process after positioning complete. |
| FIBFLM | 128,1 | |
| | | :1 LOAD(DF-EOF -1) into FIBEOF on OPEN<br>:2 Unused<br>:4 Ignore channel 12 on printer (RPG).<br>:8 DMS-II file (MCP only flag).<br>RND: Each area entered into disk directory as it is created. |
| FIBIX2 | 129,1 | IX2 Flag: Breakout Flags |
| | | All files:<br>:1 Directs MCP to move FIBARB to program IX2 after each READ or WRITE request (see FIBARB); generally used for buffer access files.<br>:2 Save previous breakout disk file.<br>:4 Use MTP for breakout (rerun every n records).<br>:8 Use DSK for breakout (ignored if :4 set). |
| FIBBCT | 130,8 | Block Count |
| | | All files:<br>Number of unique blocks of data read or written during file processing (value for RLOG taken from IOAT). |

| Label | Relative Location /Size | Content/Meaning |
|---|---|---|
| FIBSQ1 | 102,1 | Previous I/O Request Type (redefines FIBADR, FIBQAD) |
| | | SEQ I/O DSK:<br>Storage of value of FIBRWT for previous I/O request; set from FIBSQ2 at each request.<br>0 = Request was READ; if current request also a READ, must change pointers to access proper record.<br>1 = Request was WRITE; pointers are correct for current request. |
| FIBSQ2 | 103,1 | Current I/O Request Type (redefines FIBADR, FIBQAD) |
| | | Storage of value of FIBRWT for current I/O request.<br>0 = Request is READ; if previous request was also a READ, pointers (FIBARB, FIBCBS) still point at previous request and must be changed; otherwise, pointers are correct.<br>1 = Request is WRITE; must set FIBSQ3:2 to force physical I/O when block is completed; advance pointers to next record. |
| FIBSQ3 | 104,1 | Current I/O Request Type (redefines FIBADR, FIBQAD) |
| | | SEQ I/O DSK:<br>:1 Internal flag specifying both FIBSQ1 and FIBSQ2 indicates READ and must cycle back through READ/WRITE to access correct record; bit reset during return cycle.<br>:2 Specifies write activity on current block and when the block is exhausted, must be written.<br>MTP:<br>:4 Retry short/long records.<br>:8 Ignore short/long records. |
| FIBSQL | 105,5 | Sequential I/O Record Length (redefines FIBQAD, FIBADR) |
| | | SEQ I/O DSK: |

FIB

| Label | Relative Location /Size | Content/Meaning |
|---|---|---|
| | | Storage area for record length of last record accessed. |
| FIBDCF | 112,1 | DCM FILL Flag (redefines FIBDKB) |
| | | DCM:<br>0 = No FILL given.<br>1 = FILL initiated. |
| FIBABS | 120,6 | DCM actual Key location (redefines FIBKEY, FIBJAM |
| | | DCM (transparent):<br>Address of field containing record size for WCRT. |
| FIBJAM | 120,6 | Address of Jam/Missort USE routine (redefines FIBABS, FIBKEY |
| | | SOR:<br>Address of USE routine for handling jam/missort/EOF (SORTER 5). |
| FIBPSN | 138,5 | Redefine FIBPOS |
| | | TAPE:<br>FIBPOS as 4SN. |
| FIBNAU | 138,3 | Number of Disk Areas (redefines FIBPOS, FIBBFF, FIBFFL). |
| | | DSK (output only):<br>Number of disk areas assigned during run. Used internally during CLOSE. |
| FIBBFF | 138,1 | Buffer Status Flag (redefines FIBPOS, FIBNAU) |
| | | DCM (stream mode):<br>Program has initiated I/O request (buffer is ready for data). |
| FIBFFL | 139,1 | Stream Mode Flip Flag (redefines FIBPOS, FIBNAU). |
| | | DCM (stream mode):<br>Specifies operation is WCRC with stream. |
| FIBDCO | 140,1 | Stream Mode OP Storage (redefines FIBPOS, FIBNAU) |
| | | Holds low order digit of DCM OP code for current request. |
| FIBWTF | 141,1 | Wait Flag (redefines FIBPOS) |
| | | DCM (stream mode): |

The following descriptions are for disk.

| Label | Relative Location /Size | Content/Meaning |
|---|---|---|
| IO-ADR | 68,8 | Disk Address Next I/O |
| IO-AR# | 76,2 | Current Area Number |
| IO-FS1 | 78,2 | FPM Slots Assigned |
| IO-FS2 | 80,2 | FPM Slots In Use |
| IO-RBA | 82,7 | Remaining blocks in area |
| IO-DSK | 89,1 | OPEN Type |
| | | :8 Random<br>:8/ Sequential<br>:4 OPENed reverse<br>:2 COBOL code file<br>:1 Standard code file |
| IO-DK2 | 90,1 | |
| | | :8 Waiting address block core for file OPEN<br>:4 File declared as SHARED (SHRD DISK)<br>:2 File had breakout<br>:1 < <Available> > |
| IO-CLA | 91,1 | File Classification |
| | | :8 Private<br>:4 Information<br>:2 Public<br>:1 Free<br>:1/ Control |
| IO-PK1 | 92,1 | |
| | | :8 Base pack not resident on system<br>:4 Base pack type restricted<br>:2 Base pack type master<br>:1 Reserved |
| IO-PK2 | 93,1 | |
| | | :8 Pack overflow specified<br>:4 Waiting delayed OPEN (Pack)<br>:2 Waiting powered off in use pack<br>:1 < <Available> > |
| IO-HPT | 94,6 | Disk File Header Address |

The following descriptions are for split cylinder disk pack files.

| Label | Relative Location /Size | Content/Meaning |
|---|---|---|

Material Deleted by PCN 1090685-001

| | | |
|---|---|---|
| IO-RPA | 79,3 | Remaining Partitions in Area |
| IO-RBP | 82,7 | Remaining Blocks in Partition |

The following descriptions are for standard devices.

| Label | Relative Location /Size | Content/Meaning |
|---|---|---|
| IO-UST | 68,3 | Result Descriptor Last Status Update |
| IO-LKS | 71,1 | Unit Status |
| | | :8 Unit Saved<br>:4 Unit to be Saved<br>:2 Saved by MCP (secure or backup)<br>:1 Unit Locked |
| IO-LBL | 72,1 | |
| | | :8 Label sensed<br>:4,2,1<br>0 = Omitted label<br>1 = Burroughs standard label<br>2 = ANSI standard label<br>3 = B 6700 ANSI label<br>4 = ANSI label, current MCP<br>5 = B 3500 modified ANSI label, MCP and CP<br>6 = B 3500 modified ANSI label, MCPV<br>7 = LABEL1 installation label |
| IO-MOD | 73,1 | |
| | | :8 BINARY card input file<br>:4 READ with translation<br>:2 Status change<br>:1 ENABLE allowed for STATUS and I/O error |
| | 74,1 | <<Available>> |
| | | |

# SECTION 8
# RELATIVE AND INDEXED I/O (COBOL74 AND RPG ONLY)

The relative and indexed input/output intrinsics provide random and sequential access to the records in a mass storage (DISK or DISKPACK) file, and prevent access to unused or deleted record positions.

These intrinsics are each composed of a set of access routines that are automatically bound into any program that declares one or more files of that type.

Indexed and relative file formats differ from those of other files. Once created with indexed or relative organization, a file must always be declared to have that organization.

## RELATIVE I/O OVERVIEW

A Relative File consists of records which are identified by relative record numbers.

The file can be thought of as a serial string of record slots, each capable of holding a logical record. Each of these record slots is identified by an integer value (relative record number) greater than zero which specifies its logical position in the file. Records are stored and retrieved based on this number.

For example, record ten is the record associated with relative record number 10. It is stored in the tenth record slot, whether or not records have been written in the first through ninth record slots. (Conceptually, RELATIVE organization is similar to RANDOM, but it provides protection against the accessing of unused or deleted record positions.)

In the SEQUENTIAL access mode, records are accessed in ascending order by relative record number. Those record slots that do not contain records are skipped over.

In the RANDOM access mode, the order in which records are accessed is controlled by the programmer. The desired record is accessed by placing its relative record number in the relative key data item.

In the DYNAMIC access mode, the programmer can read either sequentially or randomly by using the appropriate form of READ statement. All other relative operations are performed randomly (the order may be sequential, but the random algorithms are used to perform the operation).

## INDEXED I/O OVERVIEW

An Indexed File consists of records, each uniquely identified by the value of one or more keys within that record. The records may be accessed in either a sequential manner or in a random manner by the value of a chosen key called the Key of Reference.

A record description includes one or more key data items, each associated with a key index. Each key index provides a logical access path to the data records according to the contents of the associated key data item within each record.

The PRIME KEY data item in the record description defines the prime key for that file. For the purposes of inserting, updating, and deleting records in a file; each record is identified solely by the value of its prime record key. This value must therefore be unique, and cannot be changed when updating the record.

Any ALTERNATE KEY data items in the record description define alternate keys for that file. The value of an alternate record key can be non-unique if the DUPLICATES phrase is specified. These keys provide alternate access paths for retrieval of records from the file.

In the SEQUENTIAL access mode, records are accessed in ascending order by key value of the Key of Reference. The order of retrieval of records using a key index having duplicate record key values is the order in which the records were written into the file.

In the RANDOM access mode, the sequence in which records are accessed is controlled by the programmer. The desired record is accessed by placing the value of one of its record keys into the appropriate record key data item before doing the IO.

In the DYNAMIC access mode, the programmer can read either sequentially or randomly by using the appropriate form of READ statement. All other indexed operations are performed randomly (the order may be sequential, but the random algorithms are used to perform the operation).

## RELATIVE FILE IMPLEMENTATION CONSIDERATIONS

Relative Files differ from conventional files in the B 2000/B 3000/B 4000 systems in that they have an additional block called a Control Block, and a BCI appended to each data block. In order to interpret this additional information, a set of compiler-provided intrinsics is bound into all programs that declare one or more Relative Files. A user-program operation on a Relative File generates a call to the appropriate intrinsic routine, and that routine accesses the file.

There are certain considerations for Relative Files:

1. A Relative File can be meaningfully accessed only if it is declared to be of RELATIVE ORGANIZATION in a language that supports Relative Files. Currently, only COBOL and RPG support this file organization.
2. The presence of all the access routines will increase the program size by approximately 18000 digits. Only those routines that are used by the program are bound into the code file, however, and they are overlayable within the user's overlay scheme.
3. The block size is larger than the simple product of the record size and the blocking factor because of the presence of the BCI at the end of each block. It is computed by first multiplying the user work area size by the blocking factor and padding the result to mod 4. Then the BCI length (14 + blocking factor) is added to this and the final result is arrived at by padding to mod 4 once again.
4. The first block in the file is a Control Block which contains additional information about the file beyond that stored in the file header. This block is created when the file is created, accessed when the file is accessed, and updated when the file is updated.
5. The size of the Control Block is 112 digits (56 bytes). Because it must fit into a single data block, this is the minimum blocksize the Relative File can have. If the blocksize would otherwise be less than this, the compiler will pad it out to this size and issue a warning. If this warning is encountered, space is being wasted in each data block and either the record size or the blocking factor should be increased in order to avoid this.
6. The presence of the Control Block means that the maximum number of records that can be stored in the file is "one blocks worth" less than otherwise. If this is significant, it should be taken into consideration when doing file size calculations.
7. A block is physically written only when it becomes necessary to read in a new block or to CLOSE the file, and the current block has been modified. Blocks are not necessarily written after every record modification.

8. A Relative File Data Block (RF Block) is present in the compiled program for each Relative File declared in the program. This block is 180 digits long and it contains information not in the FIB that is needed by the access routines to access and maintain the file.
9. Only one buffer is allocated for each Relative File. It is controlled solely by the access routines via the Buffer Control Structure (BCS) in the RF Block. Its size is the same as the blocksize as computed above plus up to 404 digits which are used for recovery purposes.
10. The MCP is told that a Relative File is a DMSII audit file for recovery purposes. This causes it to become a permanent file at OPEN time rather than at CLOSE time as it is for conventional files. It also causes the MCP EOF pointer to always point to the last block in the last allocated area.
11. Because of the difference between the Control Block EOF pointer and the MCP EOF pointer, utility programs (such as DMPALL) used to list the file will list all the garbage between the actual file EOF and the end of the area. This can be prevented by directing them to list only the number of blocks actually contained in the file.
12. If a program or system failure occurs during an update run, the Relative File being accessed may have an incorrect EOF pointer in the Control Block. This will cause the automatic file recovery package to be called by the next program that OPENs the file and the problem will be rectified.
13. Simultaneous use of a Relative File by an update program and any other program(s) is a feature that is not currently supported.

## RELATIVE FILE FORMATS

A Relative File is a single physical file of data and control information. The MCP is told that it is an unblocked random file. The blocks that make up the relative file are of two different formats.

The first block in the file is composed entirely of file control information and is called the Control Block. The second through last blocks are the Data Blocks. They differ from conventional data blocks only in that they have block control information (BCI) appended after the data records.

Each data block contains some number of record slots each of which contains either a valid or an empty data record. The number of slots is specified by the blocking factor in the user file declaration. Each slot is the same size as the user record area (work area).

Following the last data record slot in the block (on the next mod 4 boundary) is the block control information (BCI) used by the access routines to store and deblock the data records. This BCI consists of a reserved area and some presence flags. The reserved area (14 UN) is not used, but is present for compatability with the Indexed File Data Block format. This field should contain zeros.

Each record slot in the block has a corresponding presence flag (1 UN) in the BCI to indicate whether or not that slot contains a valid data record. A value of zero indicates that the slot is unused or that the record has been deleted; a value of one means that a record is present in that slot.

The Relative File Control Block contains file control information that is not in the file header that must be saved when the file is not in use. It was originally written out to the file using the information in the RF Block during the CLOSE following the creating OPEN. Subsequent OPENs move this information into the RF Block of the opening program. Subsequent CLOSEs map any changes back out to the Control Block.

The format of the Relative File Control Block is the same as the first 110 digits of the Indexed File Control Block. This has been done in order to allow the RELATIVE intrinsics to access INDEXED files – primarily for recovery purposes. Table 8-1 contains a description of the Relative File Control Block format :

**Table 8-1. Relative File Control Block Format**

| Usage | Size | Function |
|---|---|---|
| Version Info | | |
| | 2 UN | Release used to create file |
| | 2 UN | Format level of the file |
| | 5 UN | File creation date |
| | 10 UN | File creation time |
| | 5 UN | Last file update date |
| | 10 UN | Last file update time |
| Number of Keys | 2 UN | Number of key indexes<br>0 – relative file<br>> 0 – indexed file |
| Inuse Update Flag | 1 UN | Not used in Relative Files |
| Reserved | 13 UN | Reserved |
| File Description (FD) | | |
| | 8 UN | Pointer to associated FIB |
| | 4 UN | Reserved |
| | 6 UN | BCI offset into block (digits) |
| | 4 UN | Records per block |
| | 5 UN | User record size (digits) |
| | 5 UN | Block size (digits) |
| | 3 UN | MAX number of areas |
| | 9 UN | Blocks per area |
| | 8 UN | EOF block number (Logical EOF) |
| | 8 UN | MAX block number (Physical EOF) |
| Padding | 2 UN | Padding to make CTL BLK mod 4<br>(if data block size is padded<br>up to Control Block size) |

## RELATIVE FILE FIB

There is one FIB created by the host compiler for each Relative File using the information provided by the user in the file declaration. This FIB is created in the same manner as a FIB for a normal sequential file, with a few additions. These additions are as follows:

1. The Relative File FIB always has a user work area assigned to it. Its address is put into FIB-WA. This is the only interface the user has to the Relative File. The size of this user work area is the same as that specified in the user record description, but it is padded to mod 4 length.

2. The FIB has its block size (FIBMBS) set to the block size as computed above in Implementation Considerations.
3. The FIB has only one buffer attached to it. The size of this buffer will be equal to the block size as computed above in Implementation Considerations. FIB-AA and FIB-BB are set to its beginning and ending addresses. The compiler also allocates a certain amount of additional buffer space immediately behind the allocated buffer in order to allow the intrinsic to dynamically increase the block size when writing out a stopper pattern for recovery purposes.
4. The FIB will specify buffer access mode (FIB-BA:1 set). This prevents MCP access to the user work area. The access routines handle all movement of data to and from the user work area.
5. The FIB will specify random access mode (FIB-DA:1 set) regardless of the mode declared in the file declaration. This allows the intrinsic to access the Control Block whenever necessary and prevents MCP OPEN buffer fills. The actual access mode declared by the user will be contained in the RF Block.
6. The FIB will have the key address field (FIBKEY) set to the address of the ACTUAL KEY field in the RF Block.
7. The FIB has FIBIX2:1 reset and FIBRAD:8 set in order to force physical I/O, since the access routines determine when I/Os are to be done.
8. The FIB has FIBORG:1 set to indicate that this is a Relative File.
9. The FIB has FIBFLM:8 set dynamically prior to OPEN in order to make the MCP think the file is a DMSII audit file for recovery purposes. This bit is reset just prior to CLOSE.

## RELATIVE FILE DATA BLOCK (RF BLOCK)

The host compiler creates one Relative File Data Block (RF Block) in the code file for each Relative File declared in the user program. This block contains all the information not in the FIB that the access routines require to access and maintain a Relative File. It is the focal point throughout the program when accessing a Relative File. The address of the RF Block is the first parameter passed on each function call to an access routine. Table 8-2 is a description of the format of the RF Block.

Fields that are preceded by an * are filled in by the compiler.

**Table 8-2. RF Block Format**

| | Usage | Size | Function |
|---|---|---|---|
| * | File Status Pointer | 8 UN | Pointer to file status word<br>– all E means not declared |
| * | Open Type | 1 UN | Relative File open state<br>0 – input<br>1 – output<br>2 – input-output<br>9 – not open |
| * | Access Type | 1 UN | Declared access mode<br>0 – sequential<br>1 – random<br>2 – dynamic |

Table 8-2. RF Block Format (Cont)

| | Usage | Size | Function |
|---|---|---|---|
| | Last Function | 1 UN | Last function on Relative File<br>0 – unsuccessful<br>1 – open<br>2 – start<br>3 – read<br>4 – write<br>5 – rewrite<br>6 – delete<br>7 – close |
| | Current Record Info | | |
| | | 8 UN | Relative record number |
| | | 8 UN | Data block number |
| | | 4 SN | Data record slot number |
| | | 1 UN | State of current record<br>0 – undefined<br>1 – undefined<br>2 – beginning of file<br>3 – defined<br>4 – current record deleted |
| | Actual Key | 8 UN | Read/Write key for Relative file |
| | Buffer Control Structure (BCS) | 8 UN | Block presently in buffer |
| | | 1 UN | Block changed flag<br>0 – not changed<br>1 – changed |
| | | 1 UN | Reserved |
| * | | 6 UN | Buffer beginning address |
| | Version Info | | |
| | | 2 UN | Release used to create file |
| | | 2 UN | Format level of the file |
| | | 5 UN | File creation date |
| | | 10 UN | File creation time |
| | | 5 UN | Last file update date |
| | | 10 UN | Last file update time |
| | Indexed file info | | |
| | | 1 UN | Indexed file flag<br>0 – relative file<br>1 – indexed file |
| | | 1 UN | Indexed inuse update |
| | Relative Key Info | | |
| * | | 1 UN | Key type (units)<br>0 – UN<br>2 – UA |
| * | | 2 UN | Key length (in units) |
| * | | 6 UN | Key address |

Table 8-2. RF Block Format (Cont)

| | Usage | Size | Function |
|---|---|---|---|
| | Stopper Buffer Length | 5 UN | Length of buffer when stopper pattern is written (digits) |
| | Reserved | 13 UN | Reserved |
| | File Description | | |
| * | (FD) | 8 UN | Pointer to associated FIB |
| * | | 4 UN | Reserved |
| * | | 6 UN | BCI offset into block (digits) |
| * | | 4 UN | Records per block |
| * | | 5 UN | User record size (digits) |
| * | | 5 UN | Block size (digits) |
| * | | 3 UN | MAX number of areas |
| * | | 9 UN | Blocks per area |
| | | 8 UN | EOF block number (Logical EOF) |
| | | 8 UN | MAX block number (Physical EOF) |

# RELATIVE FILE ACCESS ROUTINE INTERFACES

The access routines required to access and maintain a Relative File are bound into the user program by the host compiler. Each program will have at most one set of routines regardless of the number of Relative Files declared. Each routine is reentrant and is capable of handling any number of Relative Files.

An access routine is called by the compiled program when it requires that a function be performed on a Relative File. These calls use the normal program stack and replace the READ, WRITE, OPEN and CLOSE BCTs that are used for conventional files. They are of the NTR-EXT type and pass call-by-address parameters. The routines and the specific parameters required by each are outlined in the following paragraphs.

## CLOSE

The CLOSE routine is called to close a Relative File. The required parameters are :

ACON address of Pointer to RF Block (8UN)
ACON address of Exception Branch Address (6UN)
ACON address of Close Type (1UN)
- 0 – normal
- 4 – release
- 6 – lock
- 8 – purge
- C – remove

## DELETE

The DELETE routine is called to remove an existing record from a Relative File. The required parameters are :

ACON address of Pointer to RF Block (8UN)
ACON address of Exception Branch Address (6UN)

## OPEN

The OPEN routine is called to open a Relative File. The required parameters are :

ACON address of Pointer to RF Block (8UN)
ACON address of Exception Branch Address (6UN)
ACON address of Open Type (1UN)
- 0 – input
- 1 – output
- 2 – input-output

ACON address of Lock Type (1UN)
- 0 – no lock
- 4 – lock access
- 8 – lock

## READ

The READ routine is called to retrieve an existing record from a Relative File. The required parameters are:

ACON address of Pointer to RF Block (8UN)
ACON address of Exception Branch Address (6UN)
ACON address of Access Type (1UN)
- 0 – sequential, no key supplied
- 1 – random, key supplied

Access Type indicates whether or not a key phrase was present in the user READ statement.

## REWRITE

The REWRITE routine is called to replace an existing record in a Relative File with the contents of the user work area. The required parameters are:

ACON address of Pointer to RF Block (8UN)
ACON address of Exception Branch Address (6UN)

## RPG-OPEN

The RPG-OPEN routine is called only by RPG programs. It is called to open a file whose organization is unknown (RELATIVE, ISAM, or CONVENTIONAL). The required parameters are :

ACON address of Pointer to RF Block (8UN)
ACON address of Exception Branch Address (6UN)
ACON address of Open Type (1UN)
0 – input
1 – output
2 – input-output
ACON address of Lock Type (1UN)
0 – no lock
4 – lock access
8 – lock
ACON address of Access Type (1UN)
0 – sequential
1 – random
VAR address of Indexed or Relative Flag (1UN)
0 – conventional file
1 – indexed or relative file

This flag is a return parameter for the RPG program.

## START

The START routine is called to position a Relative File to a particular data record for subsequent sequential access. The required parameters are :

ACON address of Pointer to RF Block (8UN)
ACON address of Exception Branch Address (6UN)
ACON address of Start Condition (1UN)
2 – at record = actual key
4 – at first record > actual key
6 – at first record >= actual key

Start Condition indicates the logical condition that was specified in the key phrase of the user START statement. If no key phrase was specified, it contains a 2.

## WRITE

The WRITE routine is called to add a new record to a Relative File. The required parameters are :

ACON address of Pointer to RF Block (8UN)
ACON address of Exception Branch Address (6UN)

# INDEXED FILE IMPLEMENTATION CONSIDERATIONS

Indexed Files differ from conventional files in the B 2000/B 3000/B 4000 systems in that their files have an additional block called a Control Block and they have a BCI appended to the end of each block. In order to interpret this additional information, compiler-provided intrinsics are bound into all programs that declare one or more Indexed Files. A user-program operation on an Indexed File generates a call to the appropriate intrinsic routine, and that routine accesses the file.

There are certain considerations for Indexed Files:

1. An Indexed File can be meaningfully accessed only if it is declared to be of INDEXED organization in a language that supports Indexed Files. Currently, only COBOL and RPG support this file organization.
2. An Indexed File declared in a program is really a group of files: a data file and one or more key files. Outside of the code file, these files are not recognized as being related. It is the user's responsibility to specify all files in dumps, loads, and so forth.
3. Each key declared for each Indexed File causes a key file to be declared to contain the resulting key index. A FIB and associated control information is allocated in the program for each of them. The net effect of this is to increase program size by 440 digits per Indexed File key.
4. The presence of all the access routines will increase the program size by approximately 45000 digits. Only those routines that are used by the program are bound into the code file, however, and they are overlayable within the user's overlay scheme.
5. There are four buffers allocated for each indexed file. They are controlled solely by the access routines via a Buffer Control Structure (BCS) in the IF Block. The user program is not allowed to access them.
6. Only one buffer is allocated for the data file of each Indexed File. Its size is the same as the data blocksize plus up to 404 digits which are used for recovery purposes.
7. Three buffers are allocated to be shared among the key files of each Indexed File. They are approximately 4500 digits each in size, so the net effect is to increase program size by approximately 13500 digits per Indexed File.
8. The data blocksize will be larger than the simple product of the record size and the blocking factor because of the presence of the BCI at the end of each block. It is computed by first multiplying the user work area size by the blocking factor and padding the result to mod 4. Then the BCI length (14 + blocking factor) is added to this and the final result is arrived at by padding to mod 4 once again.
9. The first block of the file is a Control Block, which contains additional information about the file beyond that stored in the file header. This block is created when the file is created, accessed when the file is accessed, and updated when the file is updated.
10. The size of the Control Block is 252 digits plus 100 digits for each ALTERNATE KEY declared for the file. Because it must fit into a single data block, this is the minimum blocksize the datafile can have. If the blocksize would otherwise be less than this, the compiler will pad it up to this size and issue a warning. If this warning is encountered, space is being wasted in each data block and the file is not recoverable in the event of a failure. In order to avoid these two problems, either the record size or the blocking factor must be increased sufficiently to make the data blocksize larger than the Control Block size.
11. The presence of the Control Block means that the maximum number of records that can be stored in the file is 'one block's worth' less than otherwise. If this is significant, it should be taken into consideration when doing file size calculations.

12. An Indexed File Data Block (IF Block) is present in the compiled program for each Indexed File declared in the program. This block contains certain information needed by the access routines to access and maintain the file. Its size is 380 digits plus 100 digits for each ALTERNATE KEY declared for the file.
13. A block is physically written only when it becomes necessary to read in a new block or to close the file, and the current block has been modified. Blocks are not necessarily written after every record modification.
14. The MCP is told that the Indexed File's data file is a DMSII audit file for recovery purposes. This causes it to become a permanent file at OPEN time rather than at CLOSE time as it is for standard files. It also causes the MCP EOF pointer to always point to the last block in the last allocated area.
15. Because of the difference between the Control Block EOF pointer and the MCP EOF pointer, utility programs (such as DMPALL) used to list the data file will list all the garbage between the actual file EOF and the end of the area. This can be prevented by directing them to list only the number of blocks actually contained in the file.
16. When an Indexed File is opened IO, the in-use flag in the data file Control Block is set and the file cannot be opened again until it is successfully closed by the retrieves and returns the control information to and from this block.

# INDEXED FILE FORMATS

Each Indexed File declared in a program results in multiple physical files for data and key information. The data records are stored in one file whose name is the same as the name in the file declaration. Each declared key results in the creation of a key file to contain the associated key index. The name of each key file is composed of the first 4 characters of the data file name plus the two digit key number.

## Indexed Data File Format

An Indexed Data File is a single pyhsical file of data and control information. The MCP is told that it is an unblocked random file. The blocks that make up the Indexed Data File are of two different formats.

The first block in the file is composed entirely of file control information and is called the Control Block. The second through last blocks are the Data Blocks. They differ from conventional data blocks only in that they have block control information (the BCI) appended after the data records. The Control Block does not have a BCI.

Each Data Block contains some number of record slots each of which contains either a valid or an empty data record. The number of slots is specified by the blocking factor in the user file declaration. Each slot is the same size as the user record area (work area).

Following the last data record slot in the Data Block (on the next mod 4 boundary) is the block control information (BCI) used by the access routines to store and deblock the data records. The format of the Indexed Data File BCI is as follows:

BLOCK TYPE (1 UN).
This field indicates whether or not the Data Block has any unused record slots. The allowed values are:

2 Block has available space
3 Block is full

NEXT FREE BLOCK (8 UN).
The Data Blocks containing available space are linked together in the data file using a Last In First Out (LIFO) scheme. Each block with available space points to the block number of the next free block via this field. The last free block contains zeros, as do full Data Blocks.

RECORD COUNT (4 UN).
The access routines use this field to keep a count of the number of valid records in the block as an easy way of telling when full blocks become partially filled and vice versa.

PRESENCE FLAGS (1 UN each).
Each record slot in the block has a corresponding presence flag to indicate whether or not that slot contains a valid data record. The allowed values are:

0 Record not used or was deleted
1 Record is present

The Indexed Data File Control Block contains file control information that is not in the file header that must be saved when the file is not in use. It was originally written out to the file using the information in the IF Block during the CLOSE following the creating OPEN. Subsequent OPENs move this information into the IF Block of the opening program. Subsequent CLOSEs map any changes back out to the Control Block.

The Indexed Data File Control Block also contains control information for all of the associated key files. Each key file has its own Control Block, but it only contains the date and time of its last update. For this reason, any references to the Indexed File Control Block are referring to the indexed data file's Control Block. The size of the Indexed File Control Block is 50 digits plus 100 digits for the data file FD entry plus 100 digits for each key file FD entry. Table 8-3 contains a description of its format.

**Table 8-3. Indexed File Control Block Format**

| Usage | Size | Function |
|---|---|---|
| Version Info | | |
| | 2 UN | Release used to create file |
| | 2 UN | Format level of the file |
| | 5 UN | File creation date |
| | 10 UN | File creation time |
| | 5 UN | Last file update date |
| | 10 UN | Last file update time |
| Number of Keys | 2 UN | Number of key indexes (min = 01) |
| Inuse Update Flag | 1 UN | Successful close by updater ?<br>0 – last updater closed OK<br>1 – last updater died prior to closing or is still using |
| Reserved | 13 UN | Reserved |
| File Descriptions (FDs) | | (One entry per file) |
| | 8 UN | Pointer to associated FIB |
| | 4 UN | Reserved |
| | 6 UN | BCI offset into block (digits) |
| | 4 UN | Records/entries per block |
| | 5 UN | Record/entry size (digits) |
| | 5 UN | Block size (digits) |
| | 3 UN | MAX number of areas |
| | 9 UN | Blocks per area |
| | 8 UN | EOF block number (Logical EOF) |
| | 8 UN | MAX block number (Physical EOF) |
| | 8 UN | First block with FREE space |
| | 8 UN | ROOT block number |
| | 8 UN | First FINE block number |
| | 2 UN | Number of ISAM levels |
| | 4 UN | Entries held during block split |
| | 1 UN | Key type (units)<br>0 – unsigned numeric (UN)<br>1 – signed numeric (SN)<br>2 – unsigned alpha (UA)<br>3 – signed alpha (SA)<br>4 – right signed numeric(RSN)<br>5 – right signed alpha (RSA) |
| | 3 UN | Key length (units, excl sign) |
| | 5 UN | Key offset into record (digits) |
| | 1 UN | Duplicate keys flag<br>0 – duplicates not allowed<br>1 – duplicates allowed |
| Padding | 2 UN | Padding to make CTL BLK mod 4<br>(if data block size is padded up to Control Block size) |

## Indexed Key File Format

An Indexed Key File is a single physical file of key data and control information. The MCP is told that it is an unblocked random file. The blocks that make up the Indexed Key File are of two different formats.

The first block in the file is composed of file control information and is called the Control Block. The second through last blocks are the Data Blocks otherwise known as KEY Tables. They are similar in structure to the Indexed Data File Blocks except that they contain key entries rather than data records and the BCI is somewhat different.

An Indexed Key File conceptually can be described as an inverted tree-like structure with different levels within it. This tree structure has one block or key table at the top level called a ROOT Table. It also has one or more tables at the lowest level called FINE Tables. The tables at any levels in between are called COARSE Tables.

Each block (key table) contains some number of key entry slots each of which contains either a valid or an empty key entry. The number of slots is computed by the host compiler by dividing the entry size in digits into 4500 which is the target block size.

A key entry consists of a key value whose length is the same as the corresponding key data item in the data file record, and an 8-digit pointer. The pointer portion of the entries in ROOT and COARSE Tables contain the block number of a table at the next lower level. The key value portion contains the highest key value of all the entries in that lower table. The pointer portion of FINE Table entries contains the relative record number (RRN) of the record slot in the data file that contains the record whose key value matches the key portion of that key entry. The highest key entry in the last table on each level is a special entry called the Omega Entry. The key portion of this entry is greater than the largest user specifyable key value. The Omega Entry is used as an upper bound during table searching.

Immediately following the last key entry slot in each key table (block) is the block control information (BCI) used by the access routines to access, manipulate and search the key entries. The format of the Indexed Key File BCI is as follows:

BLOCK TYPE (1 UN).
This field is used to indicate the level of the key table within the tree. The allowed values are:

2 empty key table (zero entries)
4 root table
5 coarse table
6 fine table
7 root-fine table

NEXT TABLE (8 UN).
All key tables on any level of the tree structure point to the next higher table on that level via this field. The highest table on a level contains zeros in this field.

ENTRY COUNT (4 UN).
The access routines use this field to keep a count of the number of valid key entries in the table for ease of manipulation.

START SLOT (4 UN).

Entries in each key table are packed together in a group, but the group may start in any entry slot. This field indicates the slot number (zero relative) of the slot that contains the first valid key entry in the table.

The Indexed Key File Control Block contains only version information. Its size is 16 digits and its format is as follows :

LAST UPDATE DATE (5 UN).

This field contains the date that the key file was last opened IO. It and the Last Update Time are checked against the corresponding fields in the Indexed Data File Control Block at OPEN time to insure that the versions of the data and key files all match.

LAST UPDATE TIME (10 UN).

This field contains the time that the key file was last opened IO.

RESERVED (1 UN).

## INDEXED DATA FILE FIB

There is one FIB created for the data file of each Indexed File group. This FIB is generated by using the information provided by the user in the File Control, and FD sections of the COBOL-74 program. This FIB is created in the same manner as a FIB for a normal sequential file, with a few additions. These additions are as follows:

1. The Indexed Data File FIB always has a user work area assigned to it. Its address is put into FIB-WA. This is the only interface the user has to the Indexed File. The size of this user work area is the same as that specified in the user record description, but it is padded to mod 4 length.
2. The FIB has its block size (FIBMBS) set to the block size as computed above in Implementation Considerations.
3. The FIB has only one buffer attached it. The size of this buffer is equal to the block size as computed above in Implementation Considerations. FIB-AA and FIB-BB are set to its beginning and ending addresses. The compiler also allocates a certain amount of additional buffer space immediately behind the allocated buffer in order to allow the intrinsic to dynamically increase the block size when writing out a stopper pattern for recovery purposes.
4. The FIB specifies buffer access mode (FIB-BA:1 set). This prevents MCP access to the user work area. The access routines handle all the movement of data to and from the user work area.
5. The FIB specifies random access mode (FIB-DA:1 set) regardless of the mode declared in the file declaration. This allows the intrinsic to access the Control Block whenever necessary and prevents MCP OPEN buffer fills. The actual access mode declared by the user is contained in the IF Block.
6. The FIB has the key address field (FIBKEY) set to the address of the ACTUAL KEY field in the IF Block.
7. The FIB has FIBIX2:1 reset and FIBRAD:8 set in order to force physical I/O, since the access routines determine when I/Os are to be done.
8. The FIB has FIBORG:4 set to indicate that this is the data file of an Indexed File group.
9. The FIB has FIBFLM:8 set dynamically prior to OPEN in order to make the MCP treat the file as a DMSII audit file for recovery purposes. This bit is reset just prior to CLOSE.

## INDEXED KEY FILE FIBs

The host compiler creates one FIB for each key that is declared for an Indexed File. This is necessary because each set of key values is contained in a separate physical file. These FIBs are generated based on the following requirements:

1. Key File FIBs never have a user work area assigned to them. Key values to be used in I/O functions are always placed in the user work area of the data file prior to keyed function calls.
2. Each key file's block size (table size) is computed by the host compiler based on the number of key entries that will fit into a block size of 4500 digits. This number is a target block size to prevent excessive memory requirements for the user program. If the number of key entries per block would have exceeded 300, then the block will be limited to 300 entries to prevent search times on key blocks from becoming excessive. Likewise, if the number of key entries per block would have been less than 10, the block will be forced to 10 entries to prevent an excessive number of levels in the tree structure from occurring. The block size is then computed by taking the number of key entries per block times the key entry size. To this total the host compiler adds the length of the KEY BCI (18 digits) and then pads to mod 4 length.
3. Key File FIBs have their block size (FIBMBS) set to the block size as computed in step 2 above.
4. The area size for each key file is computed by the host compiler by taking the maximum number of records in the data file and dividing by the number of key entries per block as computed in step 2 above. This number represents the number of FINE Tables needed in the key file. The number of blocks in the lowest COARSE level is obtained by dividing the number of FINE Blocks by the entries per block value. This process continues for each level of the tree structure until the divide results in one block on a level. The number of blocks on each level are then added to give the total number of blocks in the key file. One more is added for the Control Block, and the result is divided by the number of areas allocated to the key file. This final value is then placed in FIBRPA.
5. There are three key file buffers allocated to each Indexed File declared in the user program. These buffers are not permanently attached to the Key File FIBs but are shared among the key files in an Indexed File group. This implies that the buffer size must be the maximum among all key file block sizes. The beginning address of each of the three key buffers is set in the Buffer Control Structure of the Indexed File Data Block.
6. Key File FIBs always specify buffer access mode (FIB-BA:1 set), since key files do not have work areas assigned.
7. Key File FIBs specify random access mode (FIB-DA:1 set). This prevents MCP OPEN buffer fills.
8. Key File FIBs have the key address field (FIBKEY) set to the address of the ACTUAL KEY field in the IF Block.
9. Key File FIBs have FIBIX2:1 reset and FIBRAD:8 set in order to force physical I/O, since the access routines determine when I/Os are to be done.
10. Key File FIBs have FIBORG:2 set to indicate that they are the key files of an Indexed File group.

# INDEXED FILE DATA BLOCK (IF BLOCK)

The host compiler creates one Indexed File Data Block (IF Block) in the code file for each Indexed File declared in the user program. This block contains all the information not in the FIB that the access routines require to access and maintain an Indexed File. It is the focal point throughout the program when accessing an Indexed File. The address of the IF Block is the first parameter passed on each function call to an access routine. Table 8-4 is a description of the format of the IF Block.

Fields that are preceded by an * are filled in by the compiler.

**Table 8-4. Indexed File Data Block Format**

| | Function | Size | Description |
|---|---|---|---|
| * | File Status Pointer | 8 UN | Pointer to file status word<br>– all E means not declared |
| * | OPEN Type | 1 UN | Indexed File open state<br>0 – input<br>1 – output<br>2 – input-output<br>9 – not open |
| * | Access Type | 1 UN | Declared access mode<br>0 – sequential<br>1 – random<br>2 – dynamic |
| | Last Function | 1 UN | Last function on Indexed File<br>0 – unsuccessful<br>1 – open<br>2 – start<br>3 – read<br>4 – write<br>5 – rewrite<br>6 – delete<br>7 – close |
| | Current Record Info | | |
| | | 8 UN | Relative record number |
| | | 8 UN | Data block number |
| | | 4 SN | Data record slot number |
| | | 1 UN | State of current record<br>0 – undefined<br>1 – undefined<br>2 – beginning of file<br>3 – defined<br>4 – current record deleted |
| | | 2 UN | Key of reference (key ) |
| | | 8 UN | Key table number (block ) |
| | | 4 SN | Key entry slot number |

Table 8-4. Indexed File Data Block Format (Cont)

| | Function | Size | Description |
|---|---|---|---|
| | Actual Key | 8 UN | Read/Write key for all files |
| | Buffer Control Structures (BCSes) | | (One entry per buffer) |
| | | 2 UN | File number (FD entry number) |
| | | 8 UN | Block in buffer (block ) |
| | | 1 UN | Buffer changed flag<br>0 – not changed<br>1 – changed |
| | | 3 UN | Reserved |
| * | | 6 UN | Buffer beginning address |
| | Version Info | | |
| | | 2 UN | Release used to create file |
| | | 2 UN | Format level of the file |
| | | 5 UN | File creation date |
| | | 10 UN | File creation time |
| | | 5 UN | Last file update date |
| | | 10 UN | Last file update time |
| * | Number of Keys | 2 UN | Number of Keys Defined |
| | Increasing Keys | 1 UN | Order of key file update<br>0 – decreasing key order<br>1 – increasing key order |
| | Stopper Buffer Length | 5 UN | Length of buffer when stopper pattern is written (digits) |
| | Reserved | 2 UN | Reserved |
| | File Descriptions | | (One entry per file) |
| * | (FDs) | 8 UN | Pointer to associated FIB |
| * | | 4 UN | Reserved |
| * | | 6 UN | BCI offset into block (digits) |
| * | | 4 UN | Records/entries per block |
| * | | 5 UN | Record/entry size (digits) |
| * | | 5 UN | Block size (digits) |
| * | | 3 UN | MAX number of areas |
| * | | 9 UN | Blocks per area |
| | | 8 UN | EOF block number (Logical EOF) |
| | | 8 UN | MAX block number (Physical EOF) |
| | | 8 UN | First block with FREE space |
| | | 8 UN | ROOT block number |
| | | 8 UN | First FINE block number |
| | | 2 UN | Number of ISAM levels |
| * | | 4 UN | Entries held during block split |

Table 8-4. Indexed File Data Block Format (Cont)

| | Function | Size | Description |
|---|---|---|---|
| * | | 1 UN | Key type (units)<br>0 – unsigned numeric (UN)<br>1 – signed numeric (SA)<br>2 – unsigned alpha (UA)<br>3 – signed alpha (SA)<br>4 – right signed numeric(RSN)<br>5 – right signed alpha (RSA) |
| * | | 3 UN | Key length (units, excl sign) |
| * | | 5 UN | Key offset into record (digits) |
| * | | 1 UN | Duplicate keys flag<br>0 – duplicates not allowed<br>1 – duplicates allowed |

## INDEXED FILE BUFFERS

There are four buffers allocated by the host compiler for each Indexed File. The user program cannot directly access these buffers at any time and they cannot be shared with any other files. The first buffer is used exclusively for data file I/Os. The other three are shared by the key files for their I/Os. The data buffer size is determined by the blocking factor and the BCI size. The key buffers size is approximately 4500 digits. The block size for all key files is as close to this as possible, and the key buffer size is equal to the largest key file's block size.

The starting addresses of the Indexed File buffers are put into the BCS table in the IF Block by the host compiler. The data buffer address is put into element 0. The key buffer addresses are put into elements 1, 2, and 3.

## INDEXED FILES ACCESS ROUTINE INTERFACE

The access routines required to access and maintain an Indexed File are bound into the user program by the host compiler. Each program will have at most one set of routines regardless of the number of Indexed Files declared. Each routine is reentrant and is capable of handling any number of Indexed Files.

An access routine is called by the compiled program when it requires that a function be performed on an Indexed File. These calls use the normal program stack and replace the normal READ, WRITE, OPEN and CLOSE BCTs that are used for conventional files. They are of the NTR-EXT type and pass call-by-address parameters. The routines and the specific parameters required by each are outlined in the following paragraphs.

## CLOSE

The CLOSE routine is called to close an Indexed File. It updates the control blocks if needed and then closes the data file and all associated key files. The required parameters are :

ACON address of Pointer to IF Block (8UN)
ACON address of Exception Branch Address (6UN)
ACON address of Close Type (1UN)
- 0 – normal
- 4 – release
- 6 – lock
- 8 – purge
- C – remove

## DELETE

The DELETE routine is called to remove an existing record from an Indexed File. It removes the data record from the data file and the references to it from all associated key files. The required parameters are :

ACON address of Pointer to IF Block (8UN)
ACON address of Exception Branch Address (6UN)

## OPEN

The OPEN routine is called to open an Indexed File. It performs various checks and then opens the data file and all associated key files. The required parameters are :

ACON address of Pointer to IF Block (8UN)
ACON address of Exception Branch Address (6UN)
ACON address of Open Type (1UN)
- 0 – input
- 1 – output
- 2 – input-output

ACON address of Lock Type (1UN)
- 0 – no lock
- 4 – lock access
- 8 – lock

## READ

The READ routine is called to retrieve an existing data record from an Indexed File using the given key of reference. The required parameters are :

ACON address of Pointer to IF Block (8UN)
ACON address of Exception Branch Address (6UN)
ACON address of Access Type (1UN)
- 0 – sequential, no key supplied
- 1 – random, key supplied

Access Type indicates whether or not a KEY phrase was present on the user READ statement.

ACON address of Key of Reference (2UN)
00 – use current key
01 – change to prime key
02 – change to 1st alt key
03 – change to 2nd alt key
and so forth

This number indicates which key is to be used as the key of reference for this operation and those following it. It corresponds to the element number in the FD table.

ACON address of Key Address (6UN)

Key Address contains the starting address of the key data item in the user work area.

## REWRITE

The REWRITE routine is called to replace an existing record in an Indexed File with the contents of the user work area. It replaces the data record in the data file and the references to it in each key file whose associated key was changed in the new record. The required parameters are:

ACON address of Pointer to IF Block (8UN)
ACON address of Exception Branch Address (6UN)

## START

The START routine is called to position an Indexed File to a particular data record using the given key of reference for subsequent sequential access. The required parameters are:

ACON address of Pointer to IF Block (8UN)
ACON address of Exception Branch Address (6UN)
ACON address of Key of Reference (2UN)
00 – use current key
01 – change to prime key
02 – change to 1st alt key
03 – change to 2nd alt key
etc.

This number indicates which key is to be used as the key of reference at the completion of the START operation. It corresponds to the element number in the FD table.

ACON address of Key Address (6UN)

Key Address contains the starting address of the key data item in the user work area.

ACON address of Key Length (3UN)

Key Length contains the number of characters or digits (depending on key type) excluding sign that the START operation will use to compare for the specified condition. It may be less than or equal to the defined key size because the START operation will do the compare on only the number of positions specified in this parameter.

ACON address of Start Condition (1UN)
- 2 – equal
- 4 – greater
- 6 – greater than or equal

Start Condition contains the condition that the START will attempt to satisfy. This is the condition contained in the key phrase. If no key phrase was used, this field contains a 2 (equal).

## WRITE

The WRITE routine is called to add a new record to an Indexed File. It adds the data record to the data file and inserts references to it into each of the associated key files. The required parameters are:

ACON address of Pointer to IF Block (8UN)
ACON address of Exception Branch Address (6UN)

## FLOW AND DEMAND MODES OF PROCESSING

Reading of documents is accomplished either in flow or in Demand mode. In Flow mode, the Reader-Sorter reads continually until a Stop Flow is programmed or until the Reader-Sorter or the R-S DLP causes a Stop Flow to occur. In Demand mode the user program requests a single document to be fed and read.

## MEMORY MAP

The user's program must define two interface areas in the Reader-Sorter file. They are collectively known as the Memory Map. The first area is called the DLP Interface area. This area is 744 digits in length and is used for two-way communications between the user program and the DLP. The second area is called the MCP Interface area. This area is 36 digits in length and is used for two-way communications between the user program and the MCP. The total size of the Memory Map is 780 digits.

The DLP Interface area is divided into two areas. During the write portion of a Write-Flip-Read operation, the DLP transfers positions 0 – 347 from the Memory Map to the Reader-Sorter. Then during the read portion of the I/O, the DLP transfers positions 348 – 744 from its buffers to the Memory Map. Positions 0 – 347 contain pocket select, microfilm, and endorsement information. Positions 348 – 744 contain document and Result Status information.

When a Stop Flow condition occurs after a Write-Flip-Read operation is completed, the DLP will place the last used microfilm ID number in the document area (beginning at position 358) instead of document information. If the Stop Flow was caused by a jam or a missort, the DLP will also store a 2-digit code identifying the fault location of the condition on the Reader-Sorter (beginning at position 356).

The MCP Interface area is used for two-way communications with the user program. After a Reader-Sorter I/O complete, the MCP builds and stores the Soft Result Descriptor before invoking the Control State user routine. In addition, The MCP will also store status and characteristics information in the MCP Interface area when requested by a Status or Characteristics BCT. Finally, this area contains information needed by the MCP to build DLP instructions.

Figure 9-2 describes the functions of the Memory Map during a Pocket Select Read BCT. The diagram should be read from left to right.

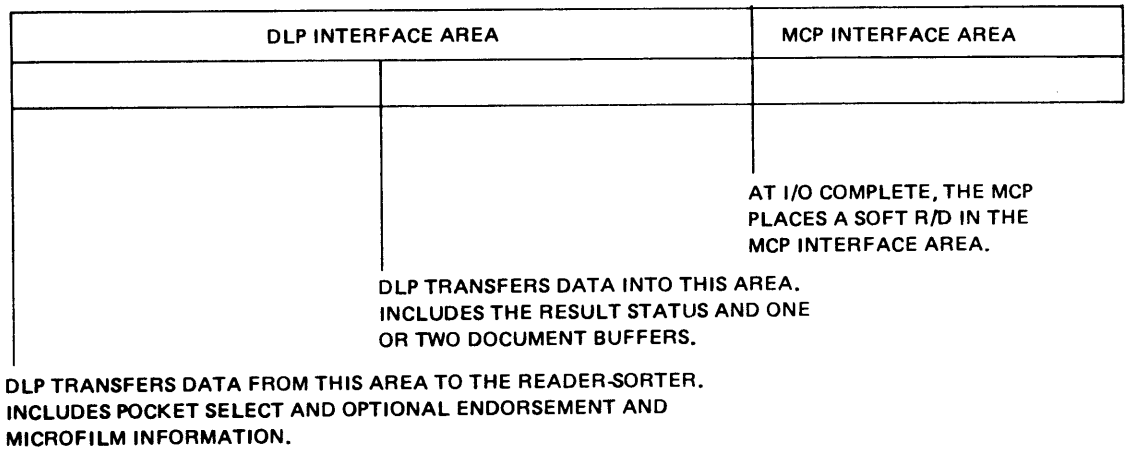



**Figure 9-2. Memory Map Functions**

The user program should zero out all reserved fields in the Memory Map before starting flow or before issuing a Demand Feed Read BCT.

The Memory Map is described Table 9-1.

**Table 9-1. Memory Map Description**

| Position | Size | Description |
|---|---|---|
| 0 | 780D | Memory Map. |
| 0 | 744D | DLP Interface area. |
| 0 | 16D | Format Delimiters for Station A. The presence of any delimiters indicates that formatting is to be done. The absence of delimiters (field filled with eight "NUL" characters) indicates that no formatting is done. If less than eight delimiters are used, then the "NUL" character must be used to complete the set. This field is used for Start Flow Reads (OP = 62) and Demand Feed Reads (OP = 63). |
| 16 | 16D | Start/Stop Pairs for Station A. A total of four pairs are allowed. The first number of each pair designates the number of increments (1/10th inch) before reading begins, while the second number of each pair denotes the number of increments before reading is stopped. All pairs are specified with reference to the right leading edge of the document. If less than four pairs are specified, "99" must be used to complete the set. If this option is not utilized, then this field must be zero-filled. This field is used for Start Flow Reads (OP = 62) and Demand Feed Reads (OP = 63). |
| 32 | 16D | Format Delimiters for Station B. Same as for Station A. |
| 48 | 16D | Start/Stop Pairs for Station B. Same as Station A. |
| 64 | 272D | Endorser bands. Refer to Non-Impact Endorsement, following, for further explanation of endorser fields. |
| 64 | 3D | Reserved. |
| 67 | 1D | Band identifier number for field 4. Must be a non-zero value (1 – 4) if this band is being loaded. |

**Table 9-1. Memory Map Description (Cont)**

| Position | Size | Description |
|---|---|---|
| 68 | 64D | Endorsement text for field 4. 32 bytes of alphanumeric endorsement text are provided. |
| 132 | 3D | Reserved. |
| 135 | 1D | Band identifier number for field 3. Must be a non-zero value (1 – 4) if this band is being loaded. |
| 136 | 64D | Endorsement text for field 3. Thirty-two bytes of alphanumeric endorsement text are provided. |
| 200 | 3D | Reserved. |
| 203 | 1D | Band identifier number for field 2. Must be a non-zero value (1 – 4) if this band is being loaded. |
| 204 | 64D | Endorsement text for field 2. 32 bytes of alphanumeric endorsement text are provided. |
| 268 | 3D | Reserved. |
| 271 | 1D | Band identifier number for field 1. Must be a non-zero value (1 – 4) if this band is being loaded. |
| 272 | 64D | Endorsement text for field 1. 32 bytes of alphanumeric endorsement text are provided. |
| 336 | 4D | Reserved. See Non-Impact Endorsement for details.<br>of this field. |
| 340 | 2D | Pocket select number. The pocket number to which the item is to be sent. If rejecting a document this number must be greater than 47. Used for Pocket Select/Read BCT (op = 29). |

**Table 9-1. Memory Map Description (Cont)**

| Position | Size | Description |
|---|---|---|
| 342 | 1D | Endorser band control.<br>8 bit = 1 spray endorser band no. 1<br>4 bit = 1 spray endorser band no. 2<br>2 bit = 1 spray endorser band no. 3<br>1 bit = 1 spray endorser band no. 4 |
| 343 | 1D | Camera/Stop Flow control.<br>8 bit = 1 microfilm this document.<br>4 bit Reserved.<br>2 bit = 1 impact endorse this document<br>1 bit = 1 Stop Flow. |
| 344 | 4D | Reserved (all zero). |
| 348 | 8D | Result Status. |
| 348 | 1D | 8 bit = 1 Station A format error. This condition occurs when document information overflows in the buffer (more than 100 bytes). This condition is reported with the affected document. (BIT 1)<br>4 bit = 1 Station A parity error from sorter. A parity error on the Reader-Sorter DLP interface was detected by the Reader-Sorter DLP. A "DLE" is used by the control to replace the character in memory. This condition is reported with the affected document. (BIT 2)<br>2 bit = 1 Station A can't read. If the sorter is unable to read a character, a "SUB" is used to replace that character. This condition is reported with the affected document. (BIT 3)<br>1 bit = 1 Station A too-late-to-read. If a Pocket Select Descriptor is received in time for pocket selection of document (N) but too late to allow reading document (N+3), then document (N) is correctly pocketed and this bit is set in the Result Status. This condition is reported with the next valid read after (N+3). Any TLTR causes the feeder and flow to stop. All documents following (N+2) are rejected. (BIT 4) |

**Table 9-1. Memory Map Description (Cont)**

| Position | Size | Description |
|---|---|---|
| 349 | 1D | 8 bit Reserved (BIT 5).<br>4 bit Reserved (BIT 6).<br>2 bit Reserved (BIT 7).<br>1 bit = 1 Document tracking error (BIT 8). |
| 350 | 1D | 8 bit = 1 Station B buffer overflow. Same as for Station A. (BIT 9)<br>4 bit = 1 Station B parity error. Same as for Station A. (R-S -> DLP) (BIT 10)<br>2 bit = 1 Station B can't read. Same as for Station A. (BIT 11)<br>1 bit Reserved (BIT 12). |
| 351 | 1D | 8 bit Reserved (BIT 13).<br>4 bit = 1 Interface error (BIT 14).<br>2 bit = 1 Internal DLP error (BIT 15).<br>1 bit Reserved (BIT 16). |
| 352 | 1D | 8 bit = 1 Flow Stopped. When true, the information following the Result Status and the jam or missort fault pocket number in memory is the last microfilm identification number (9 bytes) used by the sorter. The identification number is not meaningful unless the microfilmer is present and powered on. (BIT 17)<br>4 bit = 1 Not Ready. Any one of the following conditions exists: jam, missort, empty hopper, full pocket, start/stop bar depresed, endorser not ready, or camera not ready. (BIT 18)<br>2 bit = 1 Black Band. A black band has been detected exiting the feeder area. This is reported in conjunction with the flow-stopped bit when the black banded document has been pocket-selected. (BIT 19)<br>1 bit = 1 Endorser parity error. This condition indicates that memory transfer of endorser band data is in error. A "DLE" is used for character substitution, and this condition is reported with the next valid read. (BIT 20) |

**Table 9-1. Memory Map Description (Cont)**

| Position | Size | Description |
|---|---|---|
| | | numbers) indicate which pocket was at fault. (BIT 25) |
| | | 4 bit = 1 Feeder jam (No Feed). A jam has been detected at the feeder, the feeder has been stopped, and all unread documents are rejected. This condition is reported after flow is stopped. The "Flow Stopped" and "Not Ready" bits are set in the Result Status. (BIT 26) |
| | | 2 bit = 1 Jam. A jam has been detected at other than the feeder area. The feeder is stopped and all unread documents are rejected. This condition is reported after flow is stopped. Following the Result Status is 7 bits (2 BCD numbers) of pocket number information which tell which pocket was at fault. (BIT 27) |
| | | 1 bit = 1 Film Advance. This condition indicates that the end of a film segment was reached during document flow. The sorter goes not ready and advances the film 3 feet. (BIT 28) |
| 355 | 1D | 8 bit = 1 Reserved. (BIT 29) |
| | | 4 bit = 1 Post-read document error (mechanical). Document slippage has been detected after the first read head. The sorter rejects all fault items. This condition is reported with the next valid read. (BIT 30) |
| | | 2 bit = 1 Post read document error (electrical). The document tracking logic in the sorter was in error. The sorter rejects all fault items. This condition is reported with the next valid read. (BIT 31) |

**Table 9-1. Memory Map Description (Cont)**

| Position | Size | Description |
|---|---|---|
| | | 1 bit = 1 Parity error from the Reader-Sorter DLP. A parity error on the Reader-Sorter DLP interface was detected by the sorter. This error utilizes a "DLE" for character substitution when endorser information is being transferred to the sorter. When pocket select or other control information is in error, the Reader-Sorter DLP attempts to retransmit the information. This condition is reported with the next document. (BIT 32) |
| 356 | 20D | Flow stop information. Stored with each flow stop. |
| 356 | 2D | Fault location for jam or missort. 2-digit number specifies beam-of-light or pocket location. |
| 358 | 18D | Microfilm ID number (9 bytes). The last film ID number used by the sorter. This field is not meaningful unless the camera is powered on. |
| 356 | 388D | Document information. |
| 356 | 2D | Read station A Total document length counter. This field gives the total length of the document read at station A. Total document length is given in bytes. |
| 358 | 2D | Read station B total document length counter. Same as for station A. |
| 360 | 384D | Read Station A and B document buffers. There are two buffers each of which may contain up to 96 bytes of document information. The read station B document buffer immediately follows the last character of document information from the read station A buffer. Unless the "report feed error" option is set (B9138 utility mode) a B9138 feed check will not cause the Reader-Sorter to stop flow. The user program is notified of a feed check by a single can't read character in the document buffer. (document buffer = 01003F). |
| 744 | 36D | MCP Interface area. |

**Table 9-1. Memory Map Description (Cont)**

| Position | Size | Description |
|---|---|---|
| 744 | 8D | IOT/DLP Result Descriptor. Eight digits of R/D are always reported. The first four digits of this R/D are generated by the IOT and are referred to as the IOT R/D. The second four digits are generated by the DLP and are referred to as the DLP R/D. The MCP stores the IOT/DLP R/D in this location at each I/O Complete. |
| 752 | 4D | Reserved. |
| 756 | 1D | Number of endorser text bands to be loaded. This number must be zero through four. The DLP is capable of accepting from zero through four bands of endorsement text on each pocket select operation. (See the Non-Impact endorsement section of this document). The MCP uses this field to compute the beginning address of the "write" portion of the write flip read operation based on the number of bands to load. |
| 757 | 1D | Status. The MCP places the status of the Reader-Sorter in this area in response to a Status BCT. Formatting is as follows:<br>8 bit = 1 Slewing microfilm.<br>= 0 Not slewing microfilm.<br>4 bit = 1 Camera not ready.<br>= 0 Camera ready, not present, or not powered on.<br>2 bit = 1 Endorser not ready.<br>= 0 Endorser ready, not present, or not powered on.<br>1 bit = 1 Reader-Sorter not ready.<br>= 0 Reader-Sorter ready. |
| 758 | 3D | Characteristics. The MCP places the characteristics of its Reader-Sorter in this area in response to a Characteristics BCT. Formatting is as follows: |
| 758 | 1D | 8 bit = 1 Endorser band one present.<br>= 0 Endorser band one not present.<br>4 bit = 1 Endorser band two present.<br>= 0 Endorser band two not present.<br>2 bit = 1 Endorser band three present.<br>= 0 Endorser band three not present.<br>1 bit = 1 Endorser band four present.<br>= 0 Endorser band four not present. |

**Table 9-1. Memory Map Description (Cont)**

| Position | Size | Description |
|---|---|---|
| 759 | 1D | 8 bit = 1 Reader-Sorter is a B9137.<br>= 0 Reader-Sorter is a B9138.<br>4 bit = 1 Camera present.<br>= 0 Camera not present or powered on.<br>2 bit = 1 R-S DLP interface.<br>= 0 4A control interface.<br>1 bit = 1 Reader-Sorter power failure. |
| 760 | 1D | 8 bit = 1 Impact endorser present |
| 761 | 3D | Reserved |
| 764 | 4D | MCP Soft Result Descriptor. The MCP places a reformatted IOT/DLP result descriptor in this field when an I/O Complete of a reader-sorter BCT occurs. |

NOTE

See the "Notes on the MCP Soft Result Descriptor" for further explanation of Soft R/D error conditions.

| | | |
|---|---|---|
| 764 | 1D | 8 bit = 1 I/O invalid to the DLP (BIT 1).<br>4 bit = 1 BCT invalid to the MCP (BIT 2).<br>2 bit = 1 Flow condition error. Flow not stopped on operations requiring flow to be stopped, or flow not in process on operations requiring flow to be started. (BIT 3)<br>1 bit = 1 System interface parity error or descriptor error (BIT 4). |
| 765 | 1D | 8 bit = 1 Microfilm operation not completed. (BIT 5).<br>4 bit = 1 Non-present option required. (BIT 6).<br>2 bit = 1 Reserved (BIT 7).<br>1 bit = 1 Pocket select error (BIT 8). |
| 766 | 1D | 8 bit = 1 Bad interface information (BIT 9).<br>4 bit = 1 Timeout (BIT 10).<br>2 bit = 1 Camera not ready (BIT 11).<br>1 bit = 1 Parity error (R-S → DLP) (BIT 12). |
| 767 | 1D | 8 bit = 1 Power failure (BIT 13).<br>4 bit = 1 Memory overflow (BIT 14).<br>2 bit = 1 DLP error (BIT 15).<br>1 bit IOT R/D error (BIT 16). |

**Table 9-1. Memory Map Description (Cont)**

| Position | Size | Description |
|---|---|---|
| 353 | 1D | 8 bit = 1 Real time too late (Physical too late to pocket, endorse, or microfilm). If a pocket-select descriptor for a particular document is not received before the document reaches the time critical area, the Reader-Sorter DLP stops flow and the sorter rejects the document. This bit is set after receipt of the "Too Late" pocket-selector descriptor and is reported with the next valid read. (BIT 21) |
| | | 4 bit = 1 Multiple Documents. The sorter has detected the overlapping of multiple documents. This condition is defined as occuring before the documents are read. The fault document is rejected and this condition is reported with the affected documents. (BIT 22) |
| | | 2 bit = 1 Overlength Document. The sorter has detected an overlength item or overlapping items. This condition is also defined as occuring before the documents are read. The fault item(s) is rejected, and this condition is reported with the affected document(s). (BIT 23) |
| | | 1 bit = 1 Underspaced Document. The sorter has detected documents too close together to allow reading and/or pocket selection of the subsequent document. Both documents are rejected by the Reader-Sorter DLP, and this condition is reported with the affected documents. (BIT 24) |
| 354 | 1D | 8 bit = 1 Missort. For a missort condition, the sorter stops the feeder, and all non-fault documents in the feed line are read and pocketed. When flow is stopped, 7 bits of pocket number information are stored behind the Result Status. The 7 bits (2 BCD |

**Table 9-1. Memory Map Description (Cont)**

| Position | Size | Description |
|---|---|---|
| 768 | 4D | Pocket light/generate image count mark parameters (NNRB). |
| 768 | 2D | NN = The pocket number for which the light should be illuminated if B = 0. If B = 1, then NN equals the number of ICMs which should be generated. |
| 770 | 1D | Reserved. |
| 771 | 1D | B = 0 Pocket light illumination. On the B9138, if the cutslips/pocket light option is set on (cutslips), a cutslip will be fed from the secondary hopper and pocketed to pocket NN. The document will not be read.<br>B = 1 ICM operation.<br>B = 2 Advance batch number for impact endorser |
| 772 | 4D | Start flow parameters (RHFF). |
| 772 | 1D | Reserved. |
| 773 | 1D | H = 1 Data in the low order nine bytes of endorser band 1 is microfilm header data.<br>H = 0 No microfilm header data present. |
| 774 | 1D | F = 1 Read data from read station A. |
| 775 | 1D | F = 1 Read data from read station B. |
| 776 | 4D | Demand read parameters (RHFF). Each digit is used in the same manner as the corresponding digit in the Start Flow parameters. Separate areas are used for Start Flow and demand read parameters in order to simplify the programming effort required to use a mixture of the two techniques. |

# MCP READER-SORTER EXTENSIONS

The MCP Reader-Sorter extentions include Open BCT, Close BCT, and the MCP MICR Module.

## OPEN BCT

When opening a DLP type Reader-Sorter file, the external file identification (device name) value in the label area is compared to the device names of the unassigned DLP units. When a match is found, the unit is assigned to the program by the MCP. If a match is not found, the MCP displays NO FILE and an IL is required.

For Assembler coded programs prior to issuing the Open BCT, the user program must set the 8 bit in the first digit of the FIB (referred to as FIBST1) to declare use of this DLP interface and require selection of a B 9137/B 9138 Reader-Sorter.

## CLOSE BCT

A DLP Reader-Sorter file must be closed with Release.

## MCP MICR MODULE

The MCP requires that the MICR module be in memory when running Reader-Sorters in an on-line mode. The MICR module is automatically loaded into memory along with the standard version of the MCP at Halt/Load time provided the MICR option is set.

The MICR module has primary responsibility for the following:

1. Reader-Sorter BCT validations.
2. Building I/O descriptors to be sent to the R-S DLP.
3. Interrogating the IOT/DLP R/D at I/O Complete.
4. Formatting the Soft R/D.
5. Reinstating the user program at the proper location after a Reader-Sorter I/O Complete or after handling a Reader-Sorter BCT.

## Reader-Sorter BCTs

All MICR BCTs conform to the following general format:

```
BCT 374 (absolute memory address 374)
BUN around (next instruction)
P1 = ACON FIB
P2 = NN
P3 = ACON error
P4 = ACON flow stopped
```

P1 is a pointer to the user program sorter file FIB.

P2 is a 2-digit number which uniquely identifies the MICR BCT type (see Table 9-2).

P3 is not included in all MICR BCTs. It is a pointer to the error label in case the BCT does not pass the standard MICR BCT verification (see Table 9-2).

P4 is not included in all MICR BCTs. It is a pointer to the flow stop label.

**Table 9-2. Reader-Sorter BCT Types and Operations**

| Reader-Sorter BCT Type | Operation |
|---|---|
| 42 | Start Flow Read. |
| 43 | Demand Feed Read. |
| 44 | Pocket Light/Generate ICMs. |
| 45 | Microfilm Slew. |
| 47 | Status. |
| 48 | Characteristics. |
| 46 | Logical Read. |
| 29 | Pocket Select Read. |

# SECTION 10
# 4A CONTROL APPLICATION PROGRAM INTERFACE

## INTRODUCTION

A B 9137-3 or B 9138 Reader/Sorter (hereafter referred to as Sorter) is used with B 2000/B 3000/ B 4000 series systems through the 4A Reader/Sorter Control (hereafter referred to as Control). The Control interfaces to both the Sorter and the system through a set of bidirectional lines. Each of the lines has two possible logic levels: HIGH or LOW. Communication between the units is accomplished by changes in the level on a combination of lines. The logic levels of the lines can be viewed by placing a display monitor on the Control.

## COLDSTART/WARMSTART UNIT CARD

Unit card syntax is

UNIT cc/uu <device name> [ = ] <hardware mnemonic>

The device name is handled in the same manner as a data comm adapter-id, that is, a 1- through 6-character data name the first character of which must be alphabetic. The following five characters must be alphanumeric.

Hardware mnemonics for Sorters using a 4A Control are:

| Sorter | Mnemonic |
|---|---|
| B 9137-3 | S4A |
| B 9138 | S4B |

## USER FILE STATEMENT

In Assembler, the User File statement must conform to the following procedures:

1. Specify label convention by a blank or S in column 46 of the C address field.
2. Specify the external file identification (device name) in the A address field.
3. Contain SOR in columns 34 – 36.
4. Specify the work area and buffer with a blank, 0 or W in column 55.
5. Contain a label in column 58. Although the label is not used by the MCP or the 4A Control, the ASMBLR compiler will give a syntax error without the label.

For BPL syntax see B 2000/B 3000/B 4000 Series BPL Reference Manual form no. 1113735.

## MEMORY MAP

The user program has two interface areas for the Sorter file. They are collectively known as the Memory Map (see Table 10-1). The first area is called the Control Interface area. This area is 756 digits in length and is used by the Control to communicate with the user program. The second area is called the MCP Interface area. This area is 24 digits in length and is used by the MCP to pass information to the user program. The total size of the Memory Map is 780 digits.

The 4A Control is different from most other device controls in that it reads from the user program and writes to the user program during the same I/O operation. The Control Interface area is divided into two areas. The Control reads from the first area at locations 0 – 347, then during the same I/O

operation, writes into the second area at locations 348 – 755. As an example, consider N to be the current document passing before the Sorter read station. The Control reads the Pocket Select, Microfilm, and Endorser information for the previous document N – 1, writes the Result Status containing information for both documents N and N + 1, and writes the current document, N.

When the I/O operation is complete, the MCP places a Soft Result Descriptor in the MCP Interface area.

Figure 10-1 describes the functions of the Memory Map areas for a Pocket Select/Read BCT.

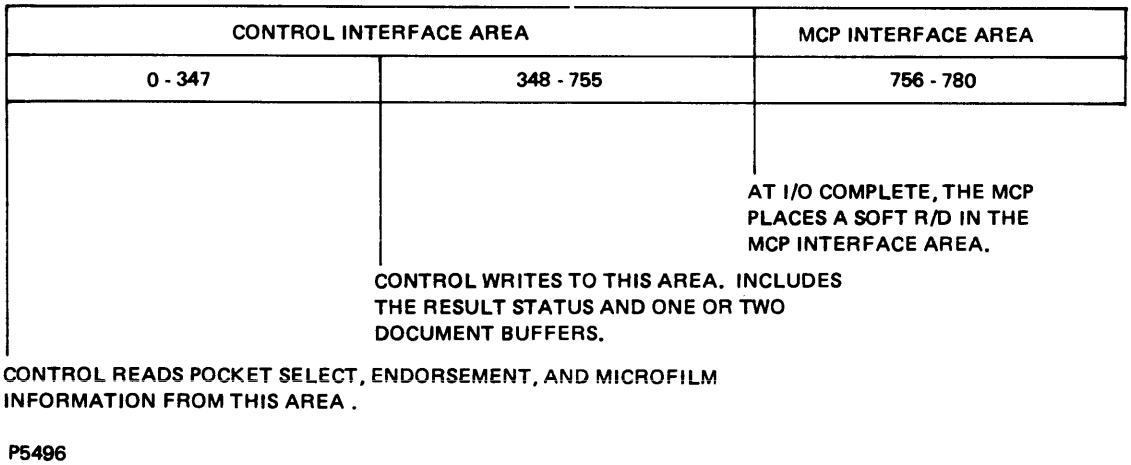



**Figure 10-1. Memory Map Functions**

NOTE

Construct the user program so as to zero out all reserved fields in the Memory Map before Start Flow begins.

The Memory Map is described in Table 10-1.

**Table 10-1. Memory Map**

| Position | Size | Description |
|---|---|---|
| 0 | 780D | Memory Map. |
| 0 | 756D | Control Interface area. |
| 0 | 16D | Format Delimiters for Station A. The presence of any delimiters indicates that formatting is to be done. The absence of delimiters (field filled with eight "NUL" characters) indicates that no formatting is done. If less than eight delimiters are used, then the "NUL" character must be used to complete the set. This field is used for Start Flow Reads (OP = 62) and Demand Feed Reads (OP = 63). |

Table 10-1. Memory Map (Cont)

| Position | Size | Description |
|---|---|---|
| 16 | 16D | Start/Stop Pairs for Station A. A total of four pairs are allowed. The first number of each pair designates the number of increments (1/10th inch) before reading begins, while the second number of each pair denotes the number of increments before reading is stopped. All pairs are specified with reference to the right leading edge of the document. If less than four pairs are specified, "99" must be used to complete the set. If this option is not utilized, then this field must be zero-filled. This field is used for Start Flow Reads (OP = 62) and Demand Feed Reads (OP = 63). |
| 32 | 16D | Format Delimiters for Station B. Same as for Station A. |
| 48 | 16D | Start/Stop Pairs for Station B. Same as for Station A. |
| 0 | 64D | Reserved. |
| 64 | 272D | Endorser Bands (Note 1). |
| 64 | 3D | Reserved. |
| 67 | 1D | Band Identifier number for field 4. Must be a value of 1,2,3 or 4 if this band is being loaded. |
| 68 | 64D | Endorsement Text for field 4. 32 bytes of alphanumeric endorsement text are provided. |
| 132 | 3D | Reserved. |
| 135 | 1D | Band Identifier number for field 3. Must be a value of 1,2,3 or 4 if this band is being loaded. |
| 136 | 64D | Endorsement Text for field 3. 32 bytes of alphanumeric endorsement text are provided. |
| 200 | 3D | Reserved. |
| 203 | 1D | Band Identifier number for field 2. Must be a value of 1,2,3 or 4 if this band is being loaded. |
| 204 | 64D | Endorsement Text for field 2. 32 bytes of alphanumeric endorsement text are provided. |
| 268 | 3D | Reserved. |

Table 10-1. Memory Map (Cont)

| Position | Size | Description |
|---|---|---|
| 271 | 1D | Band identifier number for field 1. Must be a value of 1,2,3 or 4 if this band is being loaded. |
| 272 | 64D | Endorsement text for field 1. 32 bytes of alphanumeric endorsement text are provided. |
| 336 | 4D | All zero. Indicates to Control that it is at the end of endorsement text. |
| 340 | 2D | Pocket Select number. The pocket number to which the item is to be sent. If rejecting a document this number must be greater than 47. Used for Pocket Select/Read BCT (OP = 29). |
| 342 | 1D | Endorser band control.<br>8 bit = 1 Spray Endorser Band #1.<br>4 bit = 1 Spray Endorser Band #2.<br>2 bit = 1 Spray Endorser Band #3.<br>1 bit = 1 Spray Endorser Band #4. |
| 343 | 1D | Camera/Stop Flow control.<br>8 bit = 1 microfilm this document.<br>4 bit Reserved.<br>2 bit Reserved.<br>1 bit = 1 Stop Flow. |
| 344 | 4D | Reserved. |
| 348 | 8D | Result Status. |
| 348 | 1D | 8 bit = 1 Station A format error. This condition occurs when document information overflows in the buffer (more than 100 bytes). This condition is reported with the affected document.<br>4 bit = 1 Station A parity error from sorter. A parity error on the sorter-control interface was detected by the control. A "DLE" is used by the control to replace the character in memory. This condition is reported with the affected document.<br>2 bit = 1 Station A can't read. If the sorter is unable to read a character, a "SUB" is used to replace that character. This condition is reported with the affected document. |

Table 10-1. Memory Map (Cont)

| Position | Size | Description |
|---|---|---|
| | | 1 bit = 1 Station A too-late-to-read. If a Pocket Select Descriptor is received in time for pocket selection of document (N) but too late to allow reading document (N + 3), then document (N) is correctly pocketed and this bit is set in the Result Status. This condition is reported with the next valid read after (N + 3). Any TLTR causes the feeder and flow to stop. All documents following (N + 2) are rejected. |
| 349 | 1D | 8 bit Reserved.<br>4 bit = 1 Pocket Select Error (B 9138 only).<br>2 bit Reserved.<br>1 bit Reserved. |
| 350 | 1D | 8 bit = 1 Station B Buffer Overflow. Same as for station A.<br>4 bit = 1 Station B Parity Error from Sorter. Same as for station A.<br>2 bit = 1 Station B Can't Read. Same as for station A.<br>1 bit = 1 Station B Too Late To Read. Same as for station A. |
| 351 | 1D | 8 bit Reserved.<br>4 bit Reserved.<br>2 bit Reserved.<br>1 bit Reserved. |
| 352 | 1D | 8 bit = 1 Flow Stopped. When true, the information following the Result Status and the jam or missort fault pocket number in memory is the last microfilm identification number (9 bytes) used by the sorter. The identification number is not meaningful unless the microfilmer is present and powered on.<br>4 bit = 1 Not Ready. Any one of the following conditions exists: jam, missort, empty hopper, full pocket, start/stop bar depresed, endorser not ready, or camera not ready. |

Table 10-1. Memory Map (Cont)

| Position | Size | Description |
|---|---|---|
| | | 2 bit = 1 Black Band. A black band has been detected exiting the feeder area. This is reported in conjunction with the flow-stopped bit when the black banded document has been pocket-selected. |
| | | 1 bit = 1 Failsoft Error. This condition indicates that memory transfer of endorser band data is in error. A "DLE" is used for character substitution, and this condition is reported with the next valid read. |
| 353 | 1D | 8 bit = 1 Real time too late (Physical too late to pocket, endorse, or microfilm). If a pocket-select descriptor for a particular document is not received before the document reaches the time critical area, the control stops flow and the sorter rejects the document. This bit is set after receipt of the "Too Late" pocket-select descriptor and is reported with the next valid read. |
| | | 4 bit = 1 Double Document. The sorter has detected the overlapping of multiple documents. This condition is defined as occuring before the documents are read. The fault document is rejected and this condition is reported with the affected documents. (Note 2) |
| | | 2 bit = 1 Overlength Document. The sorter has detected an overlength item or overlapping items. This condition is also defined as occuring before the documents are read. The fault item(s) is rejected, and this condition is reported with the affected document(s). (Note 2) |
| | | 1 bit = 1 Underspaced Document. The sorter has detected documents too close together to allow reading and/or pocket selection of the subsequent document. Both documents are rejected by the control, and this condition is reported with the affected documents. (Note 2) |

Table 10-1. Memory Map (Cont)

| Position | Size | Description |
|---|---|---|
| 354 | 1D | 8 bit = 1 Missort. For a missort condition, the sorter stops the feeder, and all non-fault documents in the feed line are read and pocketed. When flow is stopped, 7 bits of pocket number information are stored behind the Result Status. The 7 bits (2 BCD numbers) indicate which pocket was at fault.<br><br>4 bit = 1 Feeder jam (No Feed). A jam has been detected at the feeder, the feeder has been stopped, and all unread documents are rejected. This condition is reported after flow is stopped. The "Flow Stopped" and "Not Ready" bits are set in the Result Status. (Note 3)<br><br>2 bit = 1 Jam. A jam has been detected at other than the feeder area. The feeder is stopped and all unread documents are rejected. This condition is reported after flow is stopped. Following the Result Status is 7 bits (2 BCD numbers) of pocket number information which tell which pocket was at fault. (Note 3)<br><br>1 bit = 1 Film Advance. This condition indicates that the end of a film segment was reached during document flow. The sorter goes not ready and advances the film 3 feet. |
| 355 | 1D | 8 bit = 1 Reserved.<br><br>4 bit = 1 Post-read document error (mechanical). Document slippage has been detected after the first read head. The sorter rejects all fault items. This condition is reported with the next valid read.<br><br>2 bit = 1 Post read document error (electrical). The document tracking logic in the sorter was in error. The sorter rejects all fault items. This condition is reported with the next valid read. |

Table 10-1. Memory Map (Cont)

| Position | Size | Description |
|---|---|---|
| | | 1 bit = 1 Parity error from the control. A parity error on the sorter-control interface was detected by the sorter. This error utilizes a "DLE" for character substitution when endorser information is being transferred to the sorter. When pocket select or other control information is in error, the control attempts to retransmit the information. This condition is reported with the next document. |
| 356 | 2D | Fault location for jam or missort. 2 digit number specifies beam-of-light or pocket location. |
| 358 | 18D | Microfilm ID number(9 bytes). The last film ID number used by the Sorter. This field is not meaningful unless the camera is powered on. |
| 356 | 200D | Station A 100 byte document buffer. |
| 356 | 2D | Total document length counter. This field gives the total length of the document read at station A. The length includes ETX characters that were inserted by the Control.<br>Total document length is given in bytes. |
| 358 | 2D | Address pointer to first character. This field gives the offset, in digits, from the beginning of the document buffer to the beginning of text. This pointer is currently always 04. |
| 360 | 196D | Station A buffer. Control inserts one "ETX" character at the beginning and one "ETX" at the end of the document. If the document is an odd number of bytes, another "ETX" will be entered at the beginning of the document. Unless the "report feed error" option is set (B 9138 utility mode) a B 9138 feed check does not cause the Sorter to stop flow. The program is notified of a feed check by a single Can't Read character item (Station A Total Document buffer = 040403033F03). |
| 556 | 200D | Station B 100 byte document buffer. The format for station B is the same as for station A (Note 4). |

Table 10-1. Memory Map (Cont)

| Position | Size | Description |
|---|---|---|
| 756 | 24D | MCP Interface area. |
| 756 | 1D | Number of endorser text bands to be loaded. This number must be 0,1,2,3 or 4. The 4A Control is capable of accepting from zero through four bands of endorsement text on each Pocket Select operation. (See the Non-Impact endorsement section of this document). |
| 757 | 1D | Status. The MCP places the status of the Sorter in this area in response to a Status BCT. Formatting is as follows: |
| | | 8 bit = 1 Slewing Microfilm. |
| | | 4 bit = 1 Camera Not Ready. |
| | | = 0 Camera ready, not present, or not powered on. |
| | | 2 bit = 1 Endorser Not Ready. |
| | | = 0 Endorser ready, not present, or not powered on. |
| | | 1 bit = 1 Sorter Not Ready. |
| 758 | 2D | Characteristics. The MCP places the characteristics of the Sorter in this area in response to a Characteristics BCT. Formatting is as follows: |
| 758 | 1D | 8 bit = 1 Endorser Band one present. |
| | | 4 bit = 1 Endorser Band two present. |
| | | 2 bit = 1 Endorser Band three present. |
| | | 1 bit = 1 Endorser Band four present. |
| 759 | 1D | 8 bit = 1 Sorter is a B 9137. |
| | | = 0 Sorter is a B 9138. |
| | | 4 bit = 1 Camera present. |
| | | 2 bit Reserved. |
| | | 1 bit = 1 Read station B present. |
| 760 | 4D | Reserved. |
| 764 | 4D | MCP Soft Result Descriptor. |
| 764 | 1D | 8 bit = 1 I/O invalid to the Control. |
| | | 4 bit = 1 BCT invalid to the MCP. |

Table 10-1. Memory Map (Cont)

| Position | Size | Description |
|---|---|---|
| 772 | 4D | Start Flow parameters (RHFF). |
| 772 | 1D | Reserved. |
| 773 | 1D | H = 1 Data in the low order nine positions of endorser band 1 is microfilm header data. |
| | | H = 0 No microfilm header data present. |
| 774 | 1D | F = 1 Read data from Read Station A. |
| 775 | 1D | F = 1 Read data from Read Station B. |
| 776 | 4D | Demand Read parameters (RHFF). Each digit is used in the same manner as the corresponding digit in the Start Flow parameters. Separate areas are used for Start Flow and Demand Read parameters in order to simplify the programming effort required to use a mixture of the two techniques. |

## MCP INTERFACE (MICR 4A CONTROL BCT)

All MICR BCTs conform to the following general format:

```
BCT 374 (absolute memory address 374)
BUN around (next instruction)
P1 = ACON FIB
P2 = NN
P3 = ACON error
P4 = ACON flow stopped
```

NOTES

P1 is a pointer to the user program Sorter file FIB.

P2 is a 2-digit number which uniquely identifies the MICR BCT type (refer to Verification). For BCT Type numbers and their operations, see Table 10-2.

P3 is not included in all MICR BCTs. It is a pointer to the error label in case the BCT does not pass the standard MICR BCT verification (refer to Verification).

P4 is not included in all MICR BCT. It is a pointer to the flow stop action label.

**Table 10-2. BCT Type Numbers and Operations**

| BCT Type | Operation |
|---|---|
| 42 | Start Flow |
| 43 | Demand Feed and Read |
| 44 | Pocket Light/Generate ICM |
| 45 | Microfilm Slew |
| 47 | Status |
| 48 | Characteristics |
| 46 | Logical Read |
| 29 | Pocket Select/Read |

A detailed description of each BCT follows. Comments on Open and Close BCTs for a 4A Control Sorter file are also included.

There is a BPL syntax for the 4A Control BCT (see B 2000/B 3000/B 4000 Series BPL Reference Manual, form no. 1113735).

## STANDARD NON-POCKET SELECT/READ BCT VERIFICATION

With the exception of the Pocket Select/Read BCT which is verified in a separate routine, the Sorter BCTs are verified by the MCP in the following manner:

1. P2 must be a valid MICR BCT type. If it is not, the result is a DS or DP condition with the error message INV BCT PARAM displayed.
2. The open Sorter file table is searched by MIX # and FIB address. If an entry is not found, this results in a DS or DP condition with INV (F-N) (ADR) FILE RSTRCTD OR NOT OPEN error message displayed.
3. P3 must be non-zero, contain no undigits, be within program base/limit, and be mod 2. If not, the result is a DS or DP condition with INV BCT PARAM error message displayed.

Table 10-1. Memory Map (Cont)

| Position | Size | Description |
|---|---|---|
| | | 2 bit = 1 Flow condition error. Flow not stopped on operations requiring flow to be stopped, or flow not in process on operations requiring flow to be started. |
| | | 1 bit = 1 System interface parity error encountered on I/O initiate. |
| 765 | 1D | 8 bit = 1 Microfilm operation not completed. |
| | | 4 bit = 1 Non-present option required. |
| | | 2 bit = 1 Failsoft error encountered on control data or format, start/stop delimiters. |
| | | 1 bit = 1 Internal control Data Ram Parity error on Read Station A. |
| 766 | 1D | 8 bit = 1 Internal control Data Ram parity error on Read Station B. |
| | | 4 bit = 1 Memory Parity error. |
| | | 2 bit = 1 Failsoft error on Control to memory transfer. |
| | | 1 bit = 1 Parity error detected on Sorter to control transfer. |
| 767 | 1D | 8 bit = 1 Power failure. |
| | | 4 bit = 1 B − address exceeded. |
| | | 2 bit Reserved. |
| | | 1 bit Reserved. |
| 768 | 4D | Pocket light/generate image count mark parameters (NNRB). |
| 768 | 2D | NN = If B=0, NN is the pocket number for which the light will be illuminated. If B=1, NN equals the number of ICM that will be generated. |
| 770 | 1D | Reserved. |
| 771 | 1D | B = 0 Pocket light illumination. On the B 9138, if the cutslips/pocket light option is set on (cutslips), a cutslip will be fed from the secondary hopper and pocketed to pocket NN. The document will not be read. |
| | | B = 1 ICM operation. |

# SECTION 11
# PORT FILES

## GENERAL

The port file interface between the MCP and a user program consists of:

1. The PB-FPF4 or 5 codefile.
2. The Port FIB
3. BCTs to perform the following Port functions:

OPEN
CLOSE
READ
WRITE
GET <attribute>
SET <attribute>

4. Extensions to the event mechanism to support user-accessible Port events.

### PB-FPF = 4 OR 5 CODE FILES

Programs without Port files produce standard codefiles. When a Port file is declared in a program, PB-FPF = 4 or 5 codefiles are created. Programs without Port files produce standard codefiles. These codefiles require an MCP release level of 6.6 or greater to function properly.

A PB-FPF = 4 or 5 code file differs from the the standard code file

1. By a slight modification to the standard Program Parameter Blocks.
2. By enlarging the standard File Parameter Blocks.
3. By the addition of a Port File Parameter Block.

These differences are discussed in the following paragraphs.

### PROGRAM PARAMETER BLOCK CHANGE

Code files created with a Port file declaration must have a value of either 4 or 5 in PB-FPF, the file parameter block flag.

A value of 4 indicates that:

1. The standard File Parameter Blocks are of the 200-digit format.
2. The code file does not contain a Port File Parameter Block area.

A value of 5 indicates that:

1. The standard File Parameter Blocks are of the 200-digit format.
2. The code file contains a Port File Parameter Block area.

## FILE PARAMETER BLOCK STRUCTURE

The MCP expects file parameter blocks in the PB-FPF = 4 or 5 code file to be 200 digits long. The structure of the file parameter block is shown in Table 11-1.

**Table 11-1. File Parameter Block Structure**

| Field | Rel Loc | Size | Type | Contents |
|---|---|---|---|---|
| FP-FNM[1] | 0 | 6 | UA | Internal File Name<br>Default = 1st 6 CHAR of declared name |
| FP-MFD[1] | 12 | 6 | UA | Multi File ID<br>Default = spaces |
| FP-FID[1] | 24 | 6 | UA | File ID<br>Default = same as FP-FNM |
| FP-HWR[1] | 36 | 2 | UN | Hardware type (see FIBHDW)<br>No default, must be declared |
| FP-BUP | 38 | 1 | UN | Backup Flag<br>0 = Backup Permitted<br>:2 = Don't Backup<br>:4 = Must Backup<br>Default = 0 |
| | 39 | 1 | UN | (reserved, must = 0) |
| FP-SPF[2] | 40 | 1 | UN | Special forms flag (FIBSPF)<br>:1 = Special Forms Required<br>:2 = <Reserved><br>:4 = Auto Print<br>:8 = <Reserved><br>Default = 0 |
| FP-TRK[2] | 41 | 1 | UN | Magnetic tape track type<br>0 = Any Tape Type<br>1 = MT7<br>2 = MT9<br>3 = MPE<br>4 = GCR<br>Default = 0 |
| FP-GRD[2] | 42 | 6 | UA | Guardfile ID<br>Default = spaces |
| FP-STY[2] | 54 | 1 | UN | Security type<br>Default = @E@ |
| FP-FIB[2] | 55 | 6 | UN | FIB address (base relative)<br>No default, must be declared |
| FP-SEG[2] | 61 | 3 | UN | Segment containing FIB<br>No default, must be declared |
| FP-SUS[2] | 64 | 1 | UN | Security use<br>Default = 6 |
| | 65 | 12 | UN | (reserved, must = 0s) |
| FP-SNS | 77 | 1 | UN | Sensitive data<br>Default = 0 |

**Table 11-1. File Parameter Block Structure (Cont)**

| Field | Rel Loc | Size | Type | Contents |
|---|---|---|---|---|
| FP-FAM[1] | 78 | 6 | UA | Guardfile family name<br>Default = "DISK" |
| FP-MAI | 90 | 1 | UN | Security maintenance<br>Default = 0 |
| | 91 | 1 | UN | (reserved, must = 0s) |
| FP-YHN[1] | 92 | 17 | UA | Name of host system on which file is physically located<br>Default = spaces |
| FP-RPA | 126 | 8 | UN | Records per area<br>Default = 2000 |
| FP-AR | 134 | 4 | UN | Areas per file<br>Default = 20 |
| | 138 | 62 | UN | (reserved, must = 0s) |

NOTES

[1] Fields which must be initialized by compilers.
[2] Fields which must be initialized if known.
All other fields should also be initialized if known.

## PORT FILE PARAMETER BLOCKS

If programs contain Port files, then compilers or binders must set PB-FPF to 5, and emit a set of Port File Parameter Blocks (PFPB) in the PB-FPF=4 or 5 code file.

Although these Port File Parameter Blocks are functionally equivalent to the standard File Parameter Blocks, they are implemented differently. These PFPBs allow the MCP to obtain the initial port and subport attributes as specified in the source program, or as defaulted by the compiler. This then allows the attributes to be kept out of the executing program and provides the ability to file equate these attributes at run time.

Compilers and binders allocate code file space for the following data in a Port File Parameter Block area:

1. A directory to the port file parameter blocks.
2. Port file parameter blocks; one for each Port file declared in the source program.

### Location of PFPB Data

The PFPB directory and PFPBs immediately follow segment zero (global) in the code file. The PFPB directory starts on a sector boundary, as does each PFPB.

The code file does not contain any explicit size for this area, nor does it contain an explicit disk offset for it. The MCP determines these from other information as necessary.

## PFPB Directory

The Port File Parameter Block Directory is the first part of the PFPB area.

Because the size of the information required for a port is highly variable, the MCP needs a directory to facilitate access to the attributes of a particular port.

The Parameter Block Directory consists of contiguous entries that can span disk sector or segment boundries. The structure of this directory is given in Table 11-2.

| Field | Rel Loc | Size | Type | Contents |
|---|---|---|---|---|
| PBD-ID | 0 | 10 | UN | @FBAABDCFBE@ verify the identity of the structure. |
| PBD-LV | 10 | 2 | UN | 01 – PFPB compatibility level |
| PBD-NR | 12 | 4 | UN | number of port files declared in program |
| PBD-AD | 16 | 6 | UN | relative disk address of port file number 1 parameter [1] |
| ... | 22 | 6 | UN | relative disk address of port file number 2 parameters [1] |
| ... | ... | | | ... (one disk address entry for each port declared in PBD-NR) |

NOTE

[1] The first sector of the code file is relative disk address zero (0).

NOTE

The parameter block directory consists of contiguous entries which may span disk sector or segment boundries.

## Port File Parameter Block Structure

Compilers and binders build one Port File Parameter Block for each Port file declared in the source program (or ICMs). Each PFPB must contain at least the internal Port file name and TITLE (Port-Name) attribute, so that the Port file can be file-equated at run time. These PFPBs are located in the PFPB area, following the PFPB directory.

A Port File Parameter Block consists of variable-length strings of port and subport attributes. Each attribute consists of an attribute-number, an attribute-value-size, and an attribute-value.

Each attribute-number is four digits long and must start on a mod 2 boundary. The length of attribute-value-size is four digits, specified in the appropriate units, either digits or bytes. Digit or BOOLEAN field sizes may not exceed 99 digits.

The structure of a PFPB is port attributes, followed by zero or more optional subport attribute lists, followed by the "end-of-attributes" attribute.

Each of the optional subport attribute lists must start with a "start-of-subport" attribute. If this attribute has a value of ZERO, the subport attributes which follow apply to all subports, unless overridden.

NOTE

This "all subports" list should appear before the specific subport attribute lists to have the expected effect, since the MCP will always use the last attribute declaration when attributes are duplicated.

# PORT FILE INFORMATION BLOCK

The MCP uses a Port FIB structure to provide an interface area for the status information needed on each request, both from the program to the MCP and vice-versa.

## PORT FIB STRUCTURE

The structure of the 200-digit Port FIB is defined in Tables 11-3 through 11-5. The FIB and associated buffer must start on a mod 4 address; the minimum FIB address is 100.

**Table 11-3. Port FIB Structure**

| Field | Rel Loc | Size | Type | Contents |
|---|---|---|---|---|
| PFBVAL | 0 | 4 | UN | @CACA@ – to verify the identitiy of the structure. |
| PFBLEV | 4 | 2 | UN | 01 – compatibility level |
| PFBIOA | 6 | 6 | UN | IOAT link |
| PFBFNM | 12 | 3 | UN | file number for label equate |
| | 15 | 29 | UN | (reserved – must be 0s) |

**Table 11-4. Function Input Parameters (also includes (PFBSUB)**

| | | | | |
|---|---|---|---|---|
| PFBBSZ | 44 | 6 | UN | buffer size in bytes |
| PFBBUF | 50 | 6 | UN | buffer address (mod 4) |
| | 56 | 24 | UN | (reserved – must be 0s) |

**Table 11-5. Function Output Parameters**

| | | | | |
|---|---|---|---|---|
| PFBSUB | 80 | 4 | UN | subport index |
| PFBMSZ | 84 | 6 | UN | maximum message size in bytes |
| PFBERR | 90 | 2 | UN | error type |
| PFBSTA | 92 | 1 | UN | subport state |
| PFBEOF | 93 | 1 | UN | EOF flag on read or write |
| PFBMSG | 94 | 6 | UN | port input queue size |
| PFBINQ | 100 | 4 | UN | subport input queue size |
| PFBCUR | 104 | 6 | UN | current record size in bytes |
| PFBDAT | 110 | 5 | UN | Julian date of request |
| PFBTIM | 115 | 10 | UN | time of day in ms of request |
| | 125 | 75 | UN | (reserved – must be 0s) |

The fields in the port FIB are used as follows:

PFBVAL
Validity flag used to detect smashed FIBs. It must always be equal to @CACA@

PFBLEV
FIB design level. Currently it must be equal to 01.

PFBIOA
Link to MCP port structure while file is open. It must not be changed by the program.

PFBFNM
File number (1 relative) assigned sequentially by the compiler or binder as Port files are encountered. It is used for file equation, and it must not be changed by the program. The MCP uses this as an index into the PFPB directory. This file number is completely separate from standard file numbers.

PFBBSZ
Size in bytes of current work area. For COBOL this should be set equal to the size of the largest 01 record declared in the file on READs and set to the size of the specified 01 record on WRITEs. Other languages may change this as necessary.

PFBBUF
Address of the work area. For COBOL this will be a constant pointing to the 01s for the file, but it may vary in other languages.

PFBSUB
Subport index for this request. It must be set before each request. A value of zero means "all subports" or "port", depending on the type of request. This field will be set to the subport returning data on a successful non-directed read.

PFBMSZ
Maximum possible data size in bytes on the last I/O request.

PFBERR
Subport error value from last request. It has the following values:

0 = NO-ERROR

1 = DISCONNECTED

2 = DATA-LOST (on close)

3 = NO-BUFFER (on write no-wait)

4 = NO-FILE-FOUND (on open available)

5 = UNREACHABLE-HOST (on open)

6 = UNSUPPORTED FUNCTION (on open)

PFBSTA

Contains the subport state at the completion of the last request. It has the following values:

1 = CLOSED

2 = OPEN-PENDING

3 = OPENED

4 = BLOCKED

5 = AWAITING-HOST

6 = DEACTIVATED

7 = CLOSE-PENDING

8 = CLOSE-BLOCKED

9 = DEACTIVATION-PENDING

A = ALMOST-OPENED

B = SHUTDOWN-IN-PROCESS

C = NEVER-OPENED

PFBEOF

End-of-File flag – valid as of the completion of the last READ or WRITE request. It has the values:

0 = FALSE (no EOF detected)

1 = TRUE (EOF detected)

EOF = TRUE has several possible meanings:

1. The other program has closed its connected subfile (READ or WRITE request).
2. There is no space in the output queue (WRITE DON'T WAIT request).
3. There is no message available at this time (READ DON'T WAIT) request).
4. An interface error, such as a bad subfile index, has occurred (READ, WRITE, GET, or SET request).

See the individual function descriptions for more details.

PFBMSG

The total number of input messages remaining to be read in all subports at the completion of the last request.

PFBINQ
The number of input messages remaining to be read from the subport specified by PFBSUB at the completion of the last request.

PFBCUR
The size in bytes of the current input (or output) record as of the completion of the last read (or write) request. It cannot exceed the value in PFBBSZ.

PFBDAT
Julian date as of the completion of the last request (first half of a request timestamp).

PFBTIM
Time of day in milliseconds as of the completion of the last request (second half of a request timestamp).

## GENERATION OF PORT FIBS

Compilers or binders must build and maintain Port FIBs in user programs. Some fields in the Port FIB can be initialized at compile time in the code file. Other fields must be set up by run-time code before doing various types of run-time requests.

The following Port FIB fields must be initialized before the first run-time request:

PFBVAL
PFBLEV
PFBFNM
PFBBSZ
PFBBUF

NOTE
With PFBBSZ and PFBBUF, this depends on the compiler used.

PFBBUF does not need to be initialized if it is constant, as it is for COBOL programs. This field is constant if the buffer address is supplied as an attribute in the Port FIB for the appropriate Port. This is the preferred method of specifying a constant buffer address, because it increases the object program execution speed.

All other fields should be initialized to zeros before the first request, which can be done in the code file at compile or bind time, or dynamically at run time.

The following fields will normally be set up before each request:

PFBSUB
PFBBSZ
PFBBUF

NOTE
With PFBBSZ and PFBBUF, this depends on the compiler is use.

Compilers may make the COBOL file status field available to the user, or give the user access to the output parameters returned by the MCP after each request.

NOTE
A Port FIB must start at address 100 or above.

# PORT BCTS

The format and functional description of the various BCTs are given in the following paragraphs.

## READ REQUEST

The format of the READ Request BCT is as follows:

```
   BCT 614
   BUN <NI>
P1 ACON <Port FIB>
P2 ACON <EOF>
P3 CNST = xx   00 = wait
          02 = don't wait
P4 CNST = 10
```

This BCT requests the next message from the subport specified by PFBSUB or, if PFBSUB equals zero, from any open subport.

Depending on the state of the port and subport, either the program will have a valid message when it is reinstated, or it will be reinstated at its EOF routine.

The program will be reinstated at its EOF routine for one of the following reasons:

1. The other program is no longer connected to this subport. For example, the other program has closed its subport and all input messages have already been read. The only reasonable action for this program is to close the subport. This is a real EOF condition.
2. No message is available to be read at this time. This can happen only on a Read Don't Wait request. The subport is open and functioning normally, and a new message may come in at any time.
3. An interface error. The MCP checks some of the fields in the Port FIB, as well as the BCT parameters. Most of these interface errors will cause the MCP to terminate the program, even if it has an EOF routine.

The program determines the cause of the EOF as follows:

It is a Real EOF if either

1. The FILE-STATUS attribute is equal to DEACTIVATED.
2. PFBSTA is equal to DEACTIVATED.

No message is available if either

1. The FILE-STATUS attribute is equal to OPEN or DEACTIVATION-PENDING.
2. PFBSTA is equal to OPEN, BLOCKED, or DEACTIVATION-PENDING.

If one of the three conditions occurs when the program's EOF routine address (P2) is zero, the MCP terminates the program.

## WRITE REQUEST

The format of the WRITE Request BCT is as follows:

```
   BCT 614
   BUN <NI>
P1 ACON <Port FIB>
P2 ACON <EOF>
P3 CNST = xx    10 = wait
          11 = don't wait
P4 CNST = 10
```

This BCT requests that a message be sent through the specified subport or, if PFBSUB equals zero, through all open subports.

Depending on the state of the port and subport, either the program writes the message when it is reinstated, or it is reinstated at its EOF routine.

The program is reinstated at its EOF routine for one of the following reasons:

1. The other program is no longer connected to this subport. For example, the other program has closed its subport. This is a real EOF condition.
2. The queue size is equal to the queue limit. This can happen only on a Write Don't Wait request. The subport is open and functioning normally, and the message may be successfully rewritten by the program at a later time, when the queue size goes below the queue limit.
3. An interface error. The MCP checks some of the fields in the Port FIB, as well as the BCT parameters. Most of these interface errors will cause the MCP to terminate the program, even if it has an EOF routine.

The program determines the cause of the EOF as follows:

It is a Real EOF if

1. The FILE-STATUS attribute is equal to DEACTIVATED or DEACTIVATION-PENDING.
2. PFBSTA is deactivated or DEACTIVATION-PENDING.

A No Buffer condition exists if

1. The SUBFILE ERROR attribute is equal to NO BUFFER.
2. PFBERR is equal to NO-BUFFER.

If one of the three conditions occurs when the program's EOF routine address (P2) is zero, the MCP terminates the program.

## OPEN REQUEST

The format of the OPEN Request BCT is as follows:

```
   BCT 614
   BUN <NI>
P1 ACON <Port FIB>
P2 ACON <BASE>
P3 CNST = FE
P4 CNST = 10
P5 ACON <BASE>
P6 CNST = 20
P7 CNST = xx        00 = wait
                    10 = offer
                    20 = available
```

This request attempts to open the specified subport; if PFBSUB equals zero, it opens all supports up to the maximum number specified. The specified subport must be CLOSEd in order to be OPENed.

If error conditions arise to prevent a successful OPEN, then either the program is terminated, or the PFBERR and PFBSTA variables are set to indicate the error condition and the program is reinstated, depending on the type of error.

If no errors occur, the action taken depends on the type of OPEN (P7). These actions are:

Wait
: In this case (P6 = 20), the program is suspended until the specified subports have been successfully opened.

Available
: In this case (P6 = 2F), if no matching subport is currently available for assignment, the OPEN attempt is aborted; otherwise the OPEN is successful. Either way, the program is reinstated. PFBSTA and PFBERR must be interrogated to determine if the OPEN was successful. Two matching subports cannot become connected if both programs issue OPEN Available requests. At least one program must open its subport in such a manner that the OPEN Available function can detect that the other subport exists (that is, one program must either OPEN Wait or OPEN Offer).

Offer
: In this case (P6 = 2E), the specified subports are entered into a list of candidates for OPEN, and the program is reinstated. The actual opening of the specified subports then proceeds asynchronously with the program execution. The program can interrogate the support state of the desired support with either a GET <attribute> or COMPLEX WAIT request to determine when the specified subport becomes open. To rescind the offer of the subports, a CLOSE request must be issued for the specified subport.

## CLOSE REQUEST

The format of the CLOSE Request BCT is as follows:

```
   BCT 614
   BUN <NI>
P1 ACON <Port FIB>
P2 ACON <BASE>
P3 CNST = FF
P4 CNST = 10
P5 ACON <BASE>
P6 CNST = x     0 = retain
                4 = release
P7 CNST = x     0 = wait
                F = don't wait
P8 CNST = 00
```

This request terminates processing on the specified subport, or all subports if PFBSUB equals zero. Any input messages waiting to be read will be discarded, and a message will be sent to the connected subport which will cause all further WRITEs by the connected subport to get an EOF. After any remaining output messages have been transmitted to the connected subport, the subport is marked CLOSED and any attributes which may have changed as the result of attribute negotiations with the corresponding subport at open time resume their pre-open values. The only valid operation at this time is an OPEN.

If this is a CLOSE RETAIN request (P6 = 0), the external port structures are not deallocated and the Port FIB remains connected to them. However, the subport does not remain connected to the other process. The only use of this type of close would be to change an attribute such as YOURHOSTNAME and re-open the subport.

If this is a CLOSE RELEASE request (P6 = 4), the external subport structures are deallocated and, if this is the last open subport to be closed, the external port structures are also deallocated. After a CLOSE Release all subport attributes take their default values if an attribute interrogate or subsequent OPEN is done.

If this is a CLOSE WAIT request (P7 = 0), the program will not be reinstated unless an error is found or the CLOSE is successful.

If this is a CLOSE DON'T WAIT request (P7 = F), the program will be reinstated with either an error condition or with any CLOSE still in process. The program may detect the completion of the CLOSE by GETing state attributes or by doing a COMPLEX WAIT on the state changing.

## SET ATTRIBUTE REQUEST

The SET Attribute Request BCT has the following format.

```
   BCT 634
   BUN <NI>
P1 ACON <Port FIB>
P2 ACON <Error>
P3 CNST = FD
P4 CNST = xx   12 = pointer to other parameters.
               11 = other parameters in-line
P5 ACON <Error Results>

                                    (P4 = 11)

P6A CNST = xx   # of Attributes
    <Repeat Following Lines for Each Attribute>
P7 CNST = xxxx      CSG Attribute #
P8 CNST = xxxx Attribute Value Length
P9 ACON <Attribute Value>

                                    (P4 = 12)

P6B ACON <other parameters: P6A thru P9, as above>
                    (must be MOD 2 address)
```

This request can be used to set one or more port and subport attributes. When port attributes are to be set, PFBSUB must have a value of zero. When subport attributes are to be set, the subport number is specified by the value of PFBSUB. If PFBSUB has a value of zero, the specified subport attributes are to be set for all subports.

If an error occurs in attempting to set any attribute, the program will be reinstated at the error action label, if one exists. If no error action label is present, no indication of error will be given. An error when setting any one attribute will not prevent other valid attributes in the list from being set.

The error-results address (P5) is either zero (No Results Wanted) or points to an array of error results. This array must have one 2-digit entry for each attribute in the list. Each error result field will have one of the following values when the program is reinstated:

0 – no error
1 – invalid attribute number
2 – invalid attribute value
3 – attribute cannot ever be set
4 – attribute cannot be set while file is open
5 – invalid subport index
6 – subport not open (PLM-NS only)
7 – port not allocated
8 – invalid file type
9 – file not assigned
10 – system error

The number of attributes to be set is specified by P6A. This parameter must have a value of 01 to 99. The parameters P7 through P9 must be repeated once for each attribute specified by P6.

Each attribute to be set is specified by P7, and each is of type Numeric, Alphanumeric, Boolean, or Mnemonic. The values pointed to by P9 are assumed to be in the correct format; alphanumeric attributes are UA data, all others are UN. The attribute value length specified in P8 is assumed to be in appropriate units, UN or UA, as the case may be.

SET Attribute may be used for standard FIBs; however, the only allowable attribute is HOSTNAME. The format for the SET attribute is

```
    BCT 634
    BUN  <NI>
P1  ACON  <FIB>
P2  ACON  <Error>
P3  CNST  =  FD
P4  CNST  =  01
P5  ACON  <Error Results>
P6  CNST  =  01          # of Attributes
P7  CNST  =  0096        Attribute #
P8  CNST  =  xxxx        Attribute Value Length
P9  ACON  <Attribute Value>
```

## GET ATTRIBUTES REQUEST

The format for the GET Attribute Request BCT is as follows:

```
  BCT 654
  BUN <NI>
P1 ACON <Port FIB>
P2 ACON <Error>
P3 CNST = FC
P4 CNST = xx    12 = pointer to other parameters
          11 = other parameters in-line
P5 ACON <Error Results>

                                        (P4 = 11)

P6A CNST = xx          # of Attributes
  <Repeat Following Lines for Each Attribute>
P7 CNST = xxxx       Attribute #
P8 CNST = xxxx       Attribute Value Length
P9 ACON <Attribute Value>

                                        (P4 = 12)

P6B ACON <other parameters: P6A thru P9, as above>
          (must be MOD 2 address)
```

The GET Attribute request is used to obtain the current value of any port or subport attribute. Some port and subport attribute values are also returned in the Port FIB on the completion of each request referencing the Port FIB. Thus, it is not necessary to do a GET Attribute request for the value of subport-state following a READ request to that subport since the value is returned as a result of the READ.

The GET Attribute request must be used to obtain the values of those attributes not returned in the Port FIB, and also must be used if the attributes returned are "out of date", such as following a COMPLEX WAIT. For practical purposes, this means that in a language such as COBOL, values returned in the FIB can only be used within a basic block following a Read, Write, Open, or Close request. All other occurrences would have to generate a GET Attribute request. In other languages, such as BPL, it is the programmer's responsibility to determine which source of attribute values should be used. Syntactically this is a simple choice: one source is a reference to a subfield of the Port FIB, the other is a function call.

The syntax and pragmatics of the parameters of the GET Attribute request, and of the attributes themselves, are identical to those described for the SET Attribute request. The only exception is that GETing subport attributes with a subport index of zero is not allowed.

The GET Attribute may be used for standard FIBs; however, the only allowable attributes are AREAALLOCATED and HOSTNAME. The format is then:

```
  BCT 654
  BUN <NI>
P1 ACON <FIB>
P2 ACON <Error>
P3 CNST = FC
P4 CNST = 01
P5 ACON <Error Results>
P6 CNST = 01 # of Attributes
P7 CNST = xxxx Attribute #
P8 CNST = xxxx Attribute Value Length
P9 ACON <Attribute Value>
```

The attribute number for AREAALLOCATED is 0002 and for HOSTNAME is 0096. A GET attribute of AREAALLOCATED is valid only if the HOSTNAME attribute of the file has been set to a value which is not equal to the local hostname.

For AREAALLOCATED, the <attribute value> pointer points to a structure consisting of a 2-digit copy number (always zero) and a 4-digit area number followed by the result value. The copy number and area number fields are not included in the attribute value length.