UNITED STATES DEPARTMENT OF THE INTERIOR
GEOLOGICAL SURVEY

FORTRAN Subroutines for VAX/VMS Block I/O

Peter J. Johnson

Open-File Report 87-641

This report is preliminary and has not been reviewed for conformity with U.S. Geological Survey editorial standards and stratigraphic nomenclature. Any use of trade names is for descriptive purposes only and does not imply endorsement by the USGS.

DEC, RSX, DCL, VAX, and VMS are trademarks of Digital Equipment Corporation. WORD is a trademark of MicroSoft Corporation.

Although these subroutines have been tested by the Geological Survey, United States Department of the Interior, no warranty, either expressed, or implied is made by the Geological Survey, or by the author of this report, as to the accuracy and functionality of these routines and related material, nor shall the fact of distribution constitute any such warranty, and no responsibility is to be assumed in connection therewith.

Menlo Park, CA.

FORTRAN Subroutines for VAX/VMS Block I/O
Table of Contents

FORTRAN Subroutines for VAX/VMS Block I/O
Table of Contents

APPENDICES

Appendix A. USEROPEN Routines for QIO Block I/O

Appendix B. QIO Block I/O Routines

Appendix C. USEROPEN Routines for RMS Block I/O

Appendix D. RMS Block I/O Routines

1. Introduction

Several routines have been developed to provide a means of easily accessing various VMS block I/O facilities from FORTRAN (or any other high-level language using the same calling conventions). All routines were designed to execute 'synchronously'; that is, control is not returned to the caller until the current I/O operation has completed (this is the way standard FORTRAN I/O works). Even though some speed was sacrificed by doing things this way, the very large buffer sizes allowed (as much as 65,535 bytes for some devices, cf. VAX/VMS I/O USER'S reference manual,Part I), and the bypassing of most run-time overhead allow speed advantages over FORTRAN unformatted I/O. These speed advantages may be as much as one to two orders of magnitude, depending on the size of the reads and writes. Programmers familiar with event flag concepts may wish to gain even more speed by modifying these routines to work asynchronously. (See the RECORD MANAGEMENT SERVICES reference manual SYS$WAIT system service for modifying the RMS routines; you'll also have to set RAB$M_ASY in the RAB$L_ROP field. See the VAX/VMS SYSTEM SERVICES reference manual SYS$SYNCH system service for modifying the QIO routines to excute asynchronously.)

The other main advantage, besides speed, is that block I/O allows you to directly write any block of a sequential file, without backspacing, rewinding, etc. (this may, of course result in even more speed advantage). These routines were written to allow the replacement of as little as 1 word at any point in a file. This feature will be of obvious use to those who need to zap large data files. (In order to implement this ability, the RAB truncate-on-put bit was cleared, cf. RECORD MANAGEMENT SERVICES reference manual.)

Individual routines are documented in the next two sections. When reading the expositions on the various routines, note that the following convention is followed for arguments.

- Any argument beginning with the letter 'I' is an integer (whether 'I' is a long or a short integer is determined by the use of the /I4, or /NOI4 switch at compile-time).
- 'L' signifies a long (4-byte) integer.
- 'C' indicates that the argument is either a literal (e.g. 'filename.type', 'other.new', etc.), or a byte-string, terminated by a binary zero.
- All arguments must be passed by reference, as is usual in FORTRAN.
- Negative values of IRES are taken to be pointers to the place within the code of a subroutine, where an error occurred.

<u>References :</u>

- GUIDE TO PROGRAMMING ON VAX/VMS (FORTRAN EDITION)
- PROGRAMMING IN VAX FORTRAN
- VAX FORTRAN USER'S GUIDE
- RECORD MANAGEMENT SERVICES
- VAX/VMS SYSTEM SERVICES Reference Manual
- VAX/VMS I/O USER'S Reference Manual, Part I

2. QIO Block I/O Routines

2.1 QIO USEROPEN Routines

The routines in this section are included for purposes of completeness. They need never be called directly, if the subroutine QIO_OPEN is used for opening files.

The QIO routines do not work over DECNET. They are included because they are somewhat faster than the RMS routines, and because they will be somewhat easier to modify for asynchronous I/O (being very similar to RSX routines).

2.1.1 QIO_$OPEN

This routine is meant to be used as an argument for the FORTRAN USEROPEN keyword in an OPEN statement. It must be declared EXTERNAL and INTEGER*4 in the calling program. It is for use with the subroutines QIO_GET, QIO_PUT, QIO_OPEN, QIO_CLOSE, QIO_DELETE, QIO_TRUNCATE, and QIO_DELETE. It is for use only when opening files as 'OLD'.

2.1.2 QIO_$CREATE

This routine is similar to QIO_$OPEN, and is used for opening files 'NEW', or 'UNKNOWN'.

2.2 User-level QIO Block I/O Routines

2.2.1 QIO_OPEN(I_RW_FLAG,I_UNIT,C_FILE,L_BLOCKS,I_CHANNEL,IRES)

For those who don't want to be bothered with writing their own routines to open files, but wish to access the USEROPEN routines in section 2.1. QIO_OPEN opens all files for shared access. Options available are open for readonly, open a new file to write, or open an old (or unknown) file for write access.

I_RW_FLAG is used to indicate the desired status of the file to be opened: a value of 0 indicates readonly; 1 indicates a desire to write an old file; 2 specifies write permission for a new file; 3 indicates that an old version of the file should be opened for write access if possible, otherwise, a new file is to be created. I_UNIT specifies a FORTRAN logical unit number to be used for the file. C_FILE specifies a file name, as either a literal, or an ASCII byte-string (not as a variable of type CHARACTER). L_BLOCKS specifies an intial size in disk blocks for new files.

Values returned by this subroutine are as follows: I_CHANNEL is a VMS I/O channel number returned by the open; this value must be saved for later use by other I/O routines. IRES contains a value indicating success or failure; values greater than 0 indicate success.

Any files opened using this subroutine should be closed with the subroutine QIO_CLOSE, documented below. This routine references QIO_$CREATE and QIO_$OPEN.

2.2.2 SAVE_FIB(I_FIB)

Since QIO operations bypass the normal RMS file-handling, you may need to save the file information block, by calling SAVE_FIB immediately after QIO_OPEN, with no intervening I/O operations. This will be necessary whenever you do not know the exact size of file you will need, until after creating it; or, when you may have to truncate or delete the open file under program control. Extend (cf. QIO_EXTEND, below), truncate (cf. QIO_TRUNCATE below), and delete (cf. QIO_DELETE below) operations all require certain values to be placed in the FIB.

A contiguous array of 22 bytes is required. The address of this array is passed to SAVE_FIB in the argument I_FIB; it need not have any special structure (i.e. any array such as INTEGER*2 IFIB(11) is just fine (one per saved FIB)). Although the FIB may contain up to 48 bytes of information, the saved portion is always only 22 bytes.

2.2.3 QIO_GET(I_CHAN,I_WORDS,I_BUFFER,L_BASE,IRES)

This procedure is called to read a file, after opening it with QIO_OPEN. I_CHAN is the channel number returned by QIO_OPEN. I_WORDS is the number of 2-byte words to be read; I_WORDS is most safely thought of as a short integer, because of the limits for some devices on the size of single I/O transfers (32,367 bytes is always safe for any disk device as far as I can tell). I_BUFFER is the address of the data buffer to be read into. L_BASE is the base word (e.g. the 'base' of the first word in the file is 0, the base of the 257th word is 256, etc.) in the file, after which the read operation is to start. IRES is the return status; a value for IRES less than 0 indicates a serious error condition; a non-negative value indicates the number of words actually read. When interpreting the result of a read operation, be aware that block I/O read requests are allowed to read all the way to the end of the last block currently allocated, ignoring such niceties as the so-called end-of-file. (E.g.: if the DCL command DIRECTORY/SIZE=ALL were to return a size value of 2123/2700 for a given file, end-of-file would be 2123 but the total blocks "owned" by that file (the allocation) would be 2700.)

2.2.4 QIO_PUT(I_CHAN,I_WORDS,I_BUFFER,L_BASE,IRES)

QIO_PUT is called to write a file, after opening it with QIO_OPEN. As above, I_CHAN is the I/O channel number; I_WORDS is the number of words in the transfer; and I_BUFFER is the address of a data buffer defined in the calling procedure. L_BASE is the base-word in the file for the beginning of the write. IRES returns either a negative value, indicating failure on the write, or a positive value equal to that originally specified in I_WORDS, which indicates success.

This routine references QIO_GET. Again, as with QIO_GET, QIO_PUT will successfully write to any point in the current allocation of the file, without declaring an error.

2.2.5 QIO_EXTEND(I_CHANNEL,L_BLOCKS,I_FIB,IRES)

This routine is used to extend a disk file's allocation. Since using QIO's bypasses most of the usual run-time I/O machinery, you must extend the file's allocation any time the end-point of a put (write) operation is beyond the current file allocation. (You must extend the file before calling QIO_PUT.)

It requires as input the I/O channel (I_CHANNEL), the FIB returned by SAVE_FIB (I_FIB), and a number of blocks by which to extend the present file allocation (L_BLOCKS). It returns a result code, IRES, indicating success or failure. IRES is equal to 1 for success, and does not contain any other useful information.

2.2.6 QIO_TRUNCATE(I_CHANNEL,L_BLOCKS,I_FIB,IRES)

This operation requires as input, the I/O channel (I_CHANNEL), the total number of blocks desired in the file after truncation (L_BLOCKS), and an array for the FIB (I_FIB). IRES contains either the value 1 upon returning to the calling routine, or a negative number indicating failure of the operation. The value of L_BLOCKS must not be greater than the current last block of the file allocation.

2.2.7 QIO_DELETE(I_CHANNEL,I_UNIT,I_FIB,IRES)

This routine takes, as input, the I/O channel (I_CHANNEL), the FORTRAN logical unit (I_UNIT), and the FIB (I_FIB), which must be at least 22 bytes in length. It returns a result code (IRES), indicating success (1) or failure (0 or negative). QIO_DELETE will close and delete a file opened by QIO_OPEN. Its action is similar to a regular FORTRAN CLOSE, with STATUS='DELETE'.

This routine references QIO_CLOSE.

2.2.8 QIO_CLOSE(I_CHAN,I_UNIT,IRES)

As with standard FORTRAN I/O, files should always be properly closed. QIO_CLOSE attempts to perform a regular FORTRAN CLOSE on the logical unit (I_UNIT) specified, and disconnects the I/O stream from the I/O channnel (I_CHAN) specified. If the I/O channel cannot be successfully deassigned, an error (-1) is returned in IRES; otherwise IRES = 1.

3. RMS Block I/O Routines

These routines possess several advantages over the QIO routines: they may be used with DECNET; when the RMS block I/O routines (RMS_GET, RMS_PUT) are used, file extensions are handled automatically; and, files may be closed using the standard FORTRAN CLOSE.

3.1 RMS USEROPEN Routines

These routines may be used to open a file for synchronous block I/O. Since they invoke the full facilities of the RMS file system, the read/write operations which require them are marginally slower than the QIO routines, but probably not enough so to notice, unless critical real-time operations are involved.

The routines in this section are included for purposes of completeness. They need never be called directly, if the subroutine RMS_OPEN is used for opening files.

3.1.1 RMS_$OPEN

This routine is meant to be used as an argument for the FORTRAN USEROPEN keyword in an OPEN statement. It must be declared EXTERNAL and INTEGER*4 in the calling program. It is for use with the subroutines RMS_GET, RMS_PUT, RMS_OPEN. It is for use only when opening files as 'OLD'.

3.1.2 RMS_$CREATE

This routine is similar to RMS_$OPEN, and is used for opening files 'NEW', or 'UNKNOWN'.

3.2 User-level RMS Block I/O Routines

3.2.1 RMS_OPEN(I_RW_FLAG,I_UNIT,C_FILE,L_BLOCKS,IRES)

For those who don't want to be bothered with writing their own routines to open files, but wish to access the USEROPEN routines in section 3.1. RMS_OPEN opens all files for shared access. Options available are open for readonly, open a new file to write, or open an old (or unknown) file for write access.

I_RW_FLAG is used to indicate the desired status of the file to be opened: a value of 0 indicates readonly; 1 indicates a desire to write an old file; 2 specifies write permission for a new file; 3 indicates that an old version of the file should be opened for write access if possible, otherwise, a new file is to be created. I_UNIT specifies a FORTRAN logical unit number to be used for the file. C_FILE specifies a file name, as either a literal, or an ASCII byte-string (not as a variable of type CHARACTER). L_BLOCKS specifies an intial size in disk blocks for new files. IRES contains a value indicating success or failure; values greater than 0 indicate success.

Any files opened using this subroutine may be closed with a standard FORTRAN CLOSE statement. This routine references RMS_$CREATE and RMS_$OPEN.

3.2.2 RMS_GET(I_UNIT,I_WORDS,I_BUFFER,L_BASE,IRES)

This procedure is called to read a file, after opening it with RMS_OPEN. I_UNIT is the FORTRAN logical unit for the file. I_WORDS is the number of 2-byte words to be read; I_WORDS is most safely thought of as a short integer, because of the limits for some devices on the size of single I/O transfers (32,367 bytes is always safe for any disk device as far as I can tell). I_BUFFER is the address of the data buffer to be read into. L_BASE is the base word (e.g. the 'base' of the first word in the file is 0, the base of the 257th word is 256, etc.) in the file, after which the read operation is to start. IRES is the status return; a value for IRES less than 0 indicates a serious error condition; a non-negative value indicates the number of words actually read.

Block I/O reads will read all the way to the end-of-file block, rather than stopping at the end-of-file byte, so the returned number of words read should be interpreted with care.

3.2.3 RMS_PUT(I_UNIT,I_WORDS,I_BUFFER,L_BASE,IRES)

RMS_PUT is called to write a file, after opening it with RMS_OPEN. As above, I_UNIT is the logical unit; I_WORDS is the number of words to transfer; and I_BUFFER is the address of a data buffer defined in the calling procedure. L_BASE is the base-word in the file for the beginning of the write. IRES returns either a negative value, indicating failure on the write, or a positive value equal to that originally specified in I_WORDS, which indicates success.

Unlike QIO_PUT, RMS_PUT will automatically extend a file when doing a write (put). The truncate-on-put bit in the record access block (RAB) has been cleared by RMS_OPEN, to avoid truncating the file when the write does not take place at the end of the file. This has the side effect of only allowing a file to grow, and not to shrink.

This routine references RMS_GET.

4.0 Conclusion

The appendices contain commented FORTRAN source code for the block I/O subroutines. The code in the appendices has been edited for publication (using MicroSoft WORD v3.01) and so may contain minor typographical errors. Machine readable distribution copies of the original, running code should be available soon.

```
C ***** QIO_$OPEN : USEROPEN FORTRAN FUNCTION FOR QIO PROCESSING
C                   PASSES BACK DESCRIPTOR OF FIB WITH REQUIRED FIELDS
C                   INITIALIZED FOR DIRECTORY SEARCH(OTHERWISE, JUST
C                   DEASSIGN I/O CHANNEL TO CLOSE)

        INTEGER*4 FUNCTION QIO_$OPEN(FA_BLOCK, RA_BLOCK, LOGICAL_UNIT)

        IMPLICIT INTEGER*4 (L, S)
        BYTE CTEST

        INCLUDE '($FABDEF)'
        RECORD/FABDEF/FA_BLOCK

        INCLUDE '($RABDEF)'
        RECORD/RABDEF/RA_BLOCK

        COMMON/IO_CHANNEL/IO_CHANNEL                  ! I/O CHANNEL FOR  FILE

        COMMON/FIB_DESCRIPTOR/FIB_LENGTH, FIB_ADDRESS ! FIB DESCRIPTOR
        INTEGER*4 FIB_LENGTH, FIB_ADDRESS

        INCLUDE '($SYSSRVNAM)'

C ++ SET UFO BIT
        FA_BLOCK.FAB$L_FOP = FA_BLOCK.FAB$L_FOP .OR. FAB$M_UFO

C ++ SET FAB$V_UPI IN SHR FIELD (CF. RELEASE NOTES V4.4 P 2-19)
        FA_BLOCK.FAB$B_SHR = FA_BLOCK.FAB$B_SHR .OR. FAB$M_UPI

C ++ OPEN
        STATUS = SYS$OPEN(FA_BLOCK)
        IER = 20
        IF(.NOT.STATUS) GOTO 910

C ++ RETRIEVE CHANNEL
        IO_CHANNEL = FA_BLOCK.FAB$L_STV

C ++ SET UP FIB (FILE INFORMATION BLOCK) DESCRIPTOR,
C .. AND INITIALIZE FIELDS
        CALL INITIALIZE_FIB(%VAL(FA_BLOCK.FAB$L_NAM))

C ++ RETURN
        QIO_$OPEN = STATUS
        RETURN

C ++ ERROR MESSAGE
910     CONTINUE
        QIO_$OPEN = STATUS
        WRITE(6, 6910) IER, STATUS
6910    FORMAT(' QIO_$OPEN -- ERROR --', I7, Z12.12)
        RETURN
        END
```

```
C ***** QIO_$CREATE : USEROPEN FORTRAN FUNCTION FOR QIO PROCESSING
C           PASSES BACK DESCRIPTOR OF FIB WITH REQUIRED FIELDS
C           INITIALIZED FOR DIRECTORY SEARCH IF FILE IS TO BE EXTENDED,
C           TRUNCATED, OR DELETED ON CLOSE.

        INTEGER*4 FUNCTION QIO_$CREATE(FA_BLOCK, RA_BLOCK, LOGICAL_UNIT)

        IMPLICIT INTEGER*4 (L, S)

        INCLUDE '($FABDEF)'
        RECORD/FABDEF/FA_BLOCK

        INCLUDE '($RABDEF)'
        RECORD/RABDEF/RA_BLOCK

        COMMON/IO_CHANNEL/IO_CHANNEL                     ! I/O CHANNEL FOR FILE

        COMMON/FIB_DESCRIPTOR/FIB_LENGTH, FIB_ADDRESS    ! FIB DESCRIPTOR
        INTEGER*4 FIB_LENGTH, FIB_ADDRESS

        INCLUDE '($SYSSRVNAM)'

C ++ SET UFO BIT
        FA_BLOCK.FAB$L_FOP = FA_BLOCK.FAB$L_FOP .OR. FAB$M_UFO

C ++ SET FAB$V_UPI IN SHR FIELD (CF. RELEASE NOTES V4.4 P 2-19)
        FA_BLOCK.FAB$B_SHR = FA_BLOCK.FAB$B_SHR .OR. FAB$M_UPI

C ++ CREATE
        STATUS = SYS$CREATE(FA_BLOCK)
        IER = 20
        IF(.NOT.STATUS) GOTO 910

C ++ RETRIEVE CHANNEL
        IO_CHANNEL = FA_BLOCK.FAB$L_STV

C ++ SET UP FIB (FILE INFORMATION BLOCK) DESCRIPTOR,
C .. AND INITIALIZE FIELDS
        CALL INITIALIZE_FIB(%VAL(FA_BLOCK.FAB$L_NAM))

C ++ RETURN
        QIO_$CREATE = STATUS
        RETURN

C ++ ERROR MESSAGE
910     CONTINUE
        QIO_$CREATE = STATUS
        WRITE(6, 6910) IER, STATUS
6910    FORMAT(' QIO_$CREATE -- ERROR --', I7, Z12.12)
        RETURN
        END
```

```
C ***** INITIALIZE_FIB : INITIALIZE FIB (FILE INFORMATION BLOCK)

        SUBROUTINE INITIALIZE_FIB(NAM_BLOCK)

        INCLUDE '($NAMDEF)'
        RECORD/NAMDEF/NAM_BLOCK

        INCLUDE '($FIBDEF)'
        RECORD/FIBDEF/FI_BLOCK

        COMMON/FIB_DESCRIPTOR/FIB_LENGTH, FIB_ADDRESS   ! FIB DESCRIPTOR
        INTEGER*4 FIB_LENGTH, FIB_ADDRESS

        INCLUDE '($SYSSRVNAM)'

C ++ SPECIFY LENGTH (FIB$W_DID IS LAST FIELD USED CF. ACP QIO MANUAL)
        FIB_LENGTH = 22 ! SIZE IN BYTES :
                        ! FIB$L_ACCTL = 3
                        ! FIB$B_WSIZE = 1
                        ! FIB$W_FID   = 6
                        ! FIB$W_DID   = 6
                        ! FIB$L_WCC   = 4
                        ! FIB$W_NMCTL = 2

C ++ STUFF DESCRIPTOR WITH ADDRESS OF FIB
        FIB_ADDRESS = %LOC(FI_BLOCK)

C ++ SET WRITETHRU BIT
C .. (MARKS FILE HEADER FOR IMMEDIATE WRITE BACK TO DISK)
        FI_BLOCK.FIB$L_ACCTL = FI_BLOCK.FIB$L_ACCTL .OR. FIB$M_WRITETHRU

C ++ CLEAR DIRECTORY SEARCH CONTEXT (LONG) WORD
        FI_BLOCK.FIB$L_WCC = 0

C ++ SET TO FIND BY FILE ID
        FI_BLOCK.FIB$W_NMCTL = FI_BLOCK.FIB$W_NMCTL .OR. FIB$M_FINDFID

C ++ GET FILE ID FROM NAM BLOCK
        CALL LIB$MOVC3(6, NAM_BLOCK.NAM$W_FID, FI_BLOCK.FIB$W_FID)

C ++ GET DIRECTORY ID FROM NAM BLOCK
        CALL LIB$MOVC3(6, NAM_BLOCK.NAM$W_DID, FI_BLOCK.FIB$W_DID)

        RETURN
        END
```

```
C ***** QIO_OPEN : OPEN FILE FOR BLOCK I/O, RETURN CHANNEL

        SUBROUTINE QIO_OPEN(  I_RW_FLAG,
                              ! 0=READ (OLD)
                              ! 1=WRITE (OLD)
                              ! 2=WRITE (NEW)
                              ! 3=WRITE (UNKNOWN)
        1                     I_UNIT,     ! LOGICAL UNIT
        1                     C_FILE,     ! FILE NAME
        1                     L_BLOCKS,   ! INITIAL BLOCKS (NEW ONLY)
        1                     I_CHANNEL,  ! I/O CHANNEL
        1                     IRES)       ! RESULT

        PARAMETER W_P_B = 256             ! WORDS PER DISK BLOCK
        EXTERNAL QIO_$OPEN                ! C.F. QIOUSEROPEN.FOR
        EXTERNAL QIO_$CREATE              ! DITTO.
        COMMON/IO_CHANNEL/IO_CHANNEL      ! DITTO.

        INTEGER*4 L_BLOCKS
        BYTE C_FILE(1)

C ++ DO OPEN (DEPENDS ON VALUE OF I_RW_FLAG, I_UNIT, C_FILE)
        IRES = 0
        I_SIZE = W_P_B / 2                ! REALS PER BLOCK
        IF(I_RW_FLAG .EQ. 0) THEN         ! READ OLD FILE
                IER = 10
                OPEN(
        1       UNIT=I_UNIT,
        1       FILE=C_FILE,
        1       READONLY,
        1       SHARED,
        1       STATUS='OLD',
        1       USEROPEN=QIO_$OPEN,
        1       ERR=910)
                I_CHANNEL = IO_CHANNEL    ! RETURN CHANNEL
                IRES = I_RW_FLAG + 1      ! RESULT
        ELSE IF(I_RW_FLAG .EQ. 1) THEN    ! WRITE OLD FILE
                IER = 20
                OPEN(
        1       UNIT=I_UNIT,
        1       FILE=C_FILE,
        1       STATUS='OLD',
        1       ORGANIZATION='SEQUENTIAL',
        1       CARRIAGECONTROL='NONE',
        1       FORM='UNFORMATTED',
        1       ACCESS='SEQUENTIAL',
        1       RECORDTYPE='FIXED',
        1       RECORDSIZE=I_SIZE,
        1       USEROPEN=QIO_$OPEN,
        1       SHARED,
        1       ERR=910)
                I_CHANNEL = IO_CHANNEL    ! RETURN CHANNEL
                IRES = I_RW_FLAG + 1      ! RESULT
```

```
       ELSE IF(I_RW_FLAG .EQ. 2) THEN      ! WRITE NEW FILE
                IER = 30
                OPEN(
        1       UNIT=I_UNIT,
        1       FILE=C_FILE,
        1       STATUS='NEW',
        1       INITIALSIZE=L_BLOCKS,
        1       ORGANIZATION='SEQUENTIAL',
        1       CARRIAGECONTROL='NONE',
        1       FORM='UNFORMATTED',
        1       ACCESS='SEQUENTIAL',
        1       RECORDTYPE='FIXED',
        1       RECORDSIZE=I_SIZE,
        1       USEROPEN=QIO_$CREATE,
        1       SHARED,
        1       ERR=910)
                I_CHANNEL = IO_CHANNEL      ! RETURN CHANNEL
                IRES = I_RW_FLAG + 1        ! RESULT
        ELSE IF(I_RW_FLAG .EQ. 3) THEN      ! WRITE UNKNOWN FILE
                IER = 40
                OPEN(
        1       UNIT=I_UNIT,
        1       FILE=C_FILE,
        1       STATUS='UNKNOWN',
        1       INITIALSIZE=L_BLOCKS,
        1       ORGANIZATION='SEQUENTIAL',
        1       CARRIAGECONTROL='NONE',
        1       FORM='UNFORMATTED',
        1       ACCESS='SEQUENTIAL',
        1       RECORDTYPE='FIXED',
        1       RECORDSIZE=I_SIZE,
        1       USEROPEN=QIO_$CREATE,
        1       SHARED,
        1       ERR=910)
                I_CHANNEL = IO_CHANNEL      ! RETURN CHANNEL
                IRES = I_RW_FLAG + 1        ! RESULT
        END IF

C ++ DONE
        RETURN                              ! SUCCESS >0

C ++ ERROR ON OPEN
910    CONTINUE
       WRITE(6, 6910) IER
6910   FORMAT(' QIO_OPEN -- ERROR -- ', I6)
       IRES = -IER
       RETURN
       END
```

```
C ***** SAVE_FIB : SAVE FIB AND DESCRIPTOR,
C       IMMEDIATELY (!!!) AFTER USEROPEN
        SUBROUTINE SAVE_FIB(I_FIB)              ! USER F.I. BLOCK

        IMPLICIT INTEGER*4 (L)
        IMPLICIT BYTE (C)
        IMPLICIT REAL*8 (D)

        COMMON/FIB_DESCRIPTOR/  ! RETURNED BY QIO_$OPEN AND QIO_$CREATE
        1  FIB_LENGTH,
        1  FIB_ADDRESS
        INTEGER*4 FIB_LENGTH, FIB_ADDRESS       ! FIB_LENGTH = 22 BYTES

C ++ SAVE FIB
        CALL LIB$MOVC3(FIB_LENGTH, %VAL(FIB_ADDRESS), I_FIB)

C ++ RETURN
        RETURN
        END
```

```
C ***** QIO_GET : READ WITH QIO'S

        SUBROUTINE QIO_GET(    I_CHAN,             ! I/O CHANNEL
        1                      I_WORDS,            ! WORDS TO READ
        1                      I_BUFFER,           ! DATA BUFFER
        1                      L_BASE,             ! BASE WORD IN FILE
        1                      IRES)               ! RESULT

        IMPLICIT INTEGER*4 (L)
        INTEGER*2 I_BUFFER(1), IOSB(4)
        INTEGER*4 SYS$QIOW
        INCLUDE '($SSDEF)'
        INCLUDE '($IODEF)'
        PARAMETER W_P_B = 256                      ! WORDS PER DISK BLOCK
        INTEGER*2 I_BLOCK(W_P_B)

        LBLK1 = L_BASE/W_P_B + 1                   ! FIRST BLOCK TO READ
        I_FIRST = (W_P_B*2)                        ! READ FIRST BLOCK

        IER = 10
        L_STATUS = 0
        L_STATUS = SYS$QIOW(
        1               ,                          ! EFN
        1               %VAL(I_CHAN),              ! I/O CHANNEL
        1               %VAL(IO$_READVBLK),        ! FUNCTION
        1               IOSB,                      ! STATUS BLOCK
        1               ,                          ! AST JUNK
        1               ,                          !  "   "
        1               I_BLOCK,                   ! DATA BUFFER
        1               %VAL(I_FIRST),             ! LENGTH OF READ
        1               %VAL(LBLK1),               ! DISK BLOCK TO
                                                   ! START READING
        1               ,,)                        ! P4, P5, P6 (UNUSED)

C ++ CHECK READ ERRORS
        I_MOVE = 0
        ILEN = 0
        IF(.NOT.L_STATUS) GOTO 910
        L_STATUS = 0
        CALL LIB$MOVC3(2, IOSB(1), L_STATUS)

C ..   CHECK FOR PARTIAL READ
        IF(IOSB(1) .EQ. SS$_ENDOFFILE) THEN
            I_FIRST = 0
            L_STATUS = SS$_NORMAL
        END IF
        IER = 20
        IF(.NOT.L_STATUS) GOTO 910

C ..   IF END OF FILE FOUND ON FIRST READ, GO TO END IMMEDIATELY
        IF(I_FIRST .EQ. 0) GOTO 800
```

```
C ++ SHIFT DATA TO FIRST WORD OF BUFFER
        ISHFT = L_BASE - (LBLK1-1)*W_P_B        ! OFFSET OF USABLE DATA
        I_MOVE = I_FIRST/2 - ISHFT
        I_MOVE = MIN(I_WORDS, I_MOVE)
        CALL LIB$MOVC3(I_MOVE*2, I_BLOCK(ISHFT+1), I_BUFFER(1))
        I_GET = I_MOVE + 1

C ++ GET THE REST OF THE DATA
        ILEN = (I_WORDS-I_MOVE) * 2             ! BYTES LEFT
        IF(ILEN .LT. 0) ILEN = 0
        IF(ILEN .LE. 0) GOTO 800                ! ALL DONE

        IER = 30
        L_STATUS = 0
        LBLK = LBLK1 + 1
        L_STATUS = SYS$QIOW(
        1               ,                       ! EFN
        1               %VAL(I_CHAN),           ! I/O CHANNEL
        1               %VAL(IO$_READVBLK),     ! FUNCTION
        1               IOSB,                   ! STATUS BLOCK
        1               ,                       ! AST JUNK
        1               ,                       !  "   "
        1               I_BUFFER(I_GET),        ! DATA BUFFER
        1               %VAL(ILEN),             ! LENGTH OF READ
        1               %VAL(LBLK),             ! DISK BLOCK TO
                                                ! START READING
        1               ,,)                     ! P4, P5, P6 (UNUSED)

C ++ CHECK READ ERRORS
        IF(.NOT.L_STATUS) GOTO 910
        L_STATUS = 0
        CALL LIB$MOVC3(2, IOSB(1), L_STATUS)

C ..   CHECK FOR PARTIAL READ
        IF(IOSB(1) .EQ. SS$_ENDOFFILE) THEN
            ILEN = IOSB(2)          ! NB :  COUNT RETURNED INCLUDES
                                    ! ENTIRE EOF BLOCK
            L_STATUS = SS$_NORMAL
        END IF
        IER = 20
        IF(.NOT.L_STATUS) GOTO 910
        IER = 40
        IF(.NOT.L_STATUS) GOTO 910

C ++ ALL DONE
800     CONTINUE
        IRES = I_WORDS
        RETURN

C ++ I HATE IT WHEN THAT HAPPENS
910     CONTINUE
        WRITE(6, 6910) IER, L_STATUS
6910    FORMAT(' QIO_GET -- ERROR -- ', I6, Z10.1)
        IRES = -IER
        RETURN
        END
```

```
C ***** QIO_PUT : WRITE WITH QIO'S

        SUBROUTINE QIO_PUT(   I_CHAN,      ! I/O CHANNEL
        1                     I_WORDS,     ! WORDS TO WRITE
        1                     I_BUFFER,    ! DATA BUFFER
        1                     L_BASE,      ! BASE WORD IN FILE
        1                     IRES)        ! RESULT

        IMPLICIT INTEGER*4 (L)
        INTEGER*2 I_BUFFER(1), IOSB(4)
        INTEGER*4 SYS$QIOW
        INCLUDE '($IODEF)'
        PARAMETER W_P_B = 256              ! WORDS PER DISK BLOCK
        INTEGER*2 I_BLOCK(W_P_B)           ! ONE DISK BLOCK BUFFER


        LBLK1 = L_BASE/W_P_B + 1           ! FIRST BLOCK TO WRITE
        I_MOVE = 0                         ! WORDS MOVED TO I_BLOCK
        I_BYTES = (W_P_B*2)                ! BYTES FOR 1 BLOCK

        IF((L_BASE/W_P_B*W_P_B) .EQ. L_BASE) GOTO 50  ! WRITE STARTS ON BLOCK
                                                      ! BOUNDARY

        IER = 10
        L_STATUS = 0
        L_STATUS = SYS$QIOW(
        1               ,                        ! EFN (DEFAULT=0)
        1               %VAL(I_CHAN),            ! I/O CHANNEL
        1               %VAL(IO$_READVBLK),      ! FUNCTION
        1               IOSB,                    ! STATUS BLOCK
        1               ,                        ! AST JUNK
        1               ,                        !  "   "
        1               I_BLOCK(1),              ! DATA BUFFER
        1               %VAL(I_BYTES),           ! LENGTH OF READ
        1               %VAL(LBLK1),             ! DISK BLOCK TO READ
        1               ,,)                      ! P4,P5,P6 (UNUSED)

C ++ CHECK READ ERRORS
        IF(.NOT.L_STATUS)  GOTO 910
        L_STATUS = 0
        CALL LIB$MOVC3(2, IOSB(1), L_STATUS)
        IER = 20
        IF(.NOT.L_STATUS)  GOTO 910

C ++ SHIFT DATA FROM INPUT BUFFER
        ISHFT = L_BASE - (LBLK1-1)*W_P_B         ! OFFSET OF USABLE DATA
        I_MOVE = W_P_B - ISHFT                   ! WORDS TO MOVE FROM I_BUFFER
        I_MOVE = MIN(I_WORDS, I_MOVE)
        CALL LIB$MOVC3(I_MOVE*2, I_BUFFER, I_BLOCK(ISHFT+1))
```

```
C ++ REWRITE PARTIAL FIRST BLOCK
        IER = 30
        L_STATUS = 0
        L_STATUS = SYS$QIOW(
        1              ,
        1              %VAL(I_CHAN),
        1              %VAL(IO$_WRITEVBLK),
        1              IOSB,
        1              ,
        1              ,
        1              I_BLOCK(1),
        1              %VAL(I_BYTES),
        1              %VAL(LBLK1),
        1              ,,)

C ++ CHECK WRITE ERRORS
        IF(.NOT.L_STATUS)  GOTO 910
        L_STATUS = 0
        CALL LIB$MOVC3(2, IOSB(1), L_STATUS)
        IER = 40
        IF(.NOT.L_STATUS)  GOTO 910

C ++ WRITE MAIN BUFFER (ALL BUT LAST BLOCK)
        LBLK1 = LBLK1 + 1                    ! STARTING BLOCK TO WRITE
                                             ! DON'T INCREMENT, IF YOU
                                             ! SKIPPED FIRST PARTIAL BLOCK.
50      CONTINUE
        J_WORDS = I_WORDS - I_MOVE           ! COUNT WORDS LEFT TO WRITE
        K_WORDS = (J_WORDS/W_P_B) * W_P_B    ! ALIGN ON BLOCK BOUNDARY
        IF(K_WORDS .EQ. 0) GOTO 70           ! 1ST = LAST = PARTIAL

        K_BYTES = K_WORDS * 2                ! BYTES FOR THIS WRITE
        IER = 50
        L_STATUS = 0
        L_STATUS = SYS$QIOW(
        1              ,
        1              %VAL(I_CHAN),
        1              %VAL(IO$_WRITEVBLK),
        1              IOSB,
        1              ,
        1              ,
        1              I_BUFFER(I_MOVE+1),
        1              %VAL(K_BYTES),
        1              %VAL(LBLK1),
        1              ,,)

C ++ CHECK WRITE ERRORS
        IF(.NOT.L_STATUS)  GOTO 910
        L_STATUS = 0
        CALL LIB$MOVC3(2, IOSB(1), L_STATUS)
        IER = 60
        IF(.NOT.L_STATUS)  GOTO 910
```

```
C ++ REWRITE PARTIAL LAST BLOCK IF NECESSARY
        J_WORDS = J_WORDS - K_WORDS             ! WORDS LEFT TO WRITE
70      CONTINUE                                ! GO HERE IMMEDIATELY, WHEN
                                                ! ONLY PARTIAL BLOCKS.
        IER = 70
        IF(J_WORDS .GT. W_P_B) GOTO 910         ! S/B <= 1 BLOCK
        IF(J_WORDS .EQ. 0) GOTO 800             ! ALL DONE ... SKIP TO END

C .. GET PARTIAL LAST BLOCK
        LBLK1 = (L_BASE+I_WORDS+W_P_B-1) / W_P_B        ! LAST BLOCK TO WRITE

        IER = 80
        L_STATUS = 0
        L_STATUS = SYS$QIOW(
        1               ,
        1               %VAL(I_CHAN),
        1               %VAL(IO$_READVBLK),
        1               IOSB,
        1               ,
        1               ,
        1               I_BLOCK(1),
        1               %VAL(I_BYTES),
        1               %VAL(LBLK1),
        1               ,,)

C ++ CHECK READ ERRORS
        IF(.NOT.L_STATUS)  GOTO 910
        L_STATUS = 0
        CALL LIB$MOVC3(2, IOSB(1), L_STATUS)
        IER = 90
        IF(.NOT.L_STATUS)  GOTO 910

C ++ SHIFT DATA FROM INPUT BUFFER
        J_MOVE = I_MOVE + K_WORDS               ! WORDS ALREADY WRITTEN
        CALL LIB$MOVC3(J_WORDS*2, I_BUFFER(J_MOVE+1), I_BLOCK(1))

C ++ REWRITE PARTIAL LAST BLOCK
        IER = 100
        L_STATUS = 0
        L_STATUS = SYS$QIOW(
        1               ,
        1               %VAL(I_CHAN),
        1               %VAL(IO$_WRITEVBLK),
        1               IOSB,
        1               ,
        1               ,
        1               I_BLOCK(1),
        1               %VAL(I_BYTES),
        1               %VAL(LBLK1),
        1               ,,)

C ++ CHECK WRITE ERRORS
        IF(.NOT.L_STATUS)  GOTO 910
        L_STATUS = 0
        CALL LIB$MOVC3(2, IOSB(1), L_STATUS)
        IER = 110
        IF(.NOT.L_STATUS)  GOTO 910
```

```
C ++ FEETS, DO YOUR STUFF
800     CONTINUE
        IRES = I_WORDS
        RETURN

C ++ I HATE IT WHEN THAT HAPPENS
910     CONTINUE
        WRITE(6, 6910) IER, L_STATUS
6910    FORMAT(' QIO_PUT -- ERROR -- ', I6, Z10.1)
        IRES = -IER
        RETURN
        END
```

```
C ***** STUFF FOR FIB / ACP OPERATIONS
C          N.B.:  FIB MUST BE WORD-ALIGNED; LENGTH ALLOCATED MUST BE
C                 AT LEAST 11 WORDS (22 BYTES).

C ***** QIO_EXTEND : EXTEND FILE ALLOCATION WITH UFO BIT SET
        SUBROUTINE QIO_EXTEND(      I_CHANNEL,        ! I/O CHANNEL
        1                           L_BLOCKS,         ! EXTEND SIZE
        1                           I_FIB,            ! FIB
        1                           IRES)             ! RESULT

        IMPLICIT INTEGER*4 (L, S)
        IMPLICIT BYTE (C)
        IMPLICIT REAL*8 (D)
        INCLUDE '($IODEF)'
        INTEGER*2 IOSB(4)

        INCLUDE '($FIBDEF)'
        RECORD/FIBDEF/FI_BLOCK
        COMMON/FIB_DESCRIPTOR/FIB_LENGTH, FIB_ADDRESS
        INTEGER*4 FIB_LENGTH, FIB_ADDRESS

C ++ SET UP FIB FOR FILE EXTEND                ! CF. I/O USERS PART I
C .. TRANSFER CURRENT FIB TO COMMON
        FIB_LENGTH = 22
        CALL LIB$MOVC3(FIB_LENGTH, I_FIB, FI_BLOCK)

        FIB_ADDRESS = %LOC(FI_BLOCK)

C .. SET EXTEND, CONTIGUOUS (OR BEST TRY),
C .. ALLOCATE SPECIFIED EXTEND SIZE
        FI_BLOCK.FIB$W_EXCTL = FI_BLOCK.FIB$W_EXCTL .OR. FIB$M_EXTEND
        1                      .OR. FIB$M_ALCONB .OR. FIB$M_ALDEF

C .. SET EXTEND SIZE
        FI_BLOCK.FIB$L_EXSZ = L_BLOCKS

C .. ZERO OUT VBN OF BLOCKS ALLOCATED
        FI_BLOCK.FIB$L_EXVBN = 0

C .. ZERO OUT ALLOCATION OPTIONS
        FI_BLOCK.FIB$B_ALOPTS = 0

C .. ZERO OUT ALLOCATION ALIGNMENT SPECIFICATIONS (IGNORE REST OF FIB)
        FI_BLOCK.FIB$B_ALALIGN = 0

C .. SPECIFY LENGTH OF THIS FIB
        FIB_LENGTH = 44                   ! 44 BYTES (FIG 1-4 I/O P I)
```

```
C ++ DO EXTEND
        L_STATUS = 0
        L_STATUS = SYS$QIOW(
        1       ,                       ! DEFAULT EFN = 0
        1       %VAL(I_CHANNEL),
        1       %VAL(IO$_MODIFY),
        1       IOSB,,,
        1       FIB_LENGTH,,,,,)

        IER = 125
        IF(.NOT.L_STATUS) GOTO 910
        CALL LIB$MOVC3(2, IOSB(1), L_STATUS)
        IER = 130
        IF(.NOT.L_STATUS) GOTO 910

        IRES = 1
        RETURN

910     CONTINUE
        WRITE(6, 6910) IER, IRES, L_STATUS
6910    FORMAT(' QIO_EXTEND -- ERROR -- ', 2I7, Z10.1)
        IRES = -IER
        RETURN
        END
```

```
C ***** QIO_TRUNCATE : TRUNCATE FILE ALLOCATION WITH UFO BIT SET
C       (AND SET EOF)

        SUBROUTINE QIO_TRUNCATE(
        1               I_CHANNEL,              ! I/O CHANNEL
        1               L_BLOCKS,               ! TRUNCATE SIZE
        1               I_FIB,                  ! FIB
        1               IRES)                   ! RESULT

        IMPLICIT INTEGER*4 (L, S)
        IMPLICIT BYTE (C)
        IMPLICIT REAL*8 (D)
        INCLUDE '($IODEF)'
        INTEGER*2 IOSB(4)

        INCLUDE '($ATRDEF)/LIST'          ! ATTRIBUTE CONTROL BLOCK
        RECORD/ATRDEF/ATR_BLOCK
        INTEGER*2 IACB(6)
        INTEGER*4 LACB(2)
        EQUIVALENCE (LACB, IACB(3))
        PARAMETER W_P_B = 256             ! BLOCK SIZE (WORDS)

C ***** FATDEF : STRUCTURE DEFINITION FOR ACP RECORD ATTRIBUTE VALUES
C ***** CRIBBED FROM SYS$LIBRARY:LIB.MLB VMS V4.2 (MAY CHANGE IN FUTURE)

C .. ALLOWABLE VALUES FOR FAT$W_RTYPE
        PARAMETER       FAT$C_UNDEFINED = 0
        PARAMETER       FAT$C_FIXED     = 1
        PARAMETER       FAT$C_VARIABLE  = 2
        PARAMETER       FAT$C_VFC       = 3
        PARAMETER       FAT$C_STREAM    = 4
        PARAMETER       FAT$C_STREAMLF  = 5
        PARAMETER       FAT$C_STREAMCR  = 6

C .. ALLOWABLE VALUES FOR FAT$W_RTYPE + FAT$V_FILEORG (BIT OFFSET)
        PARAMETER       FAT$C_SEQUENTIAL= 0
        PARAMETER       FAT$C_RELATIVE  = 1
        PARAMETER       FAT$C_INDEXED   = 2
        PARAMETER       FAT$C_DIRECT    = 3

C .. MASKS FOR FAT$W_RATTRIB
        PARAMETER       FAT$M_FORTRANCC = 1
        PARAMETER       FAT$M_IMPLIEDCC = 2
        PARAMETER       FAT$M_PRINTCC   = 4
        PARAMETER       FAT$M_NOSPAN    = 8

C .. LENGTH OF FAT BLOCK
        PARAMETER       FAT$K_LENGTH    = 32
        PARAMETER       FAT$C_LENGTH    = 32
        PARAMETER       FAT$S_FATDEF    = 32
```

```
C .. LENGTH OF FAT$W_RTYPE
        PARAMETER       FAT$S_RTYPE     = 4

C .. BIT OFFSET OF FAT$W_RTYPE
        PARAMETER       FAT$V_RTYPE     = 0

C .. BIT OFFSET OF FAT$V_FILEORG
        PARAMETER       FAT$S_FILEORG   = 4
        PARAMETER       FAT$V_FILEORG   = 4

C .. BIT OFFSET OF MASKS FOR FAT$W_RATTRIB
        PARAMETER       FAT$V_FORTRANCC = 0
        PARAMETER       FAT$V_IMPLIEDCC = 1
        PARAMETER       FAT$V_PRINTCC   = 2
        PARAMETER       FAT$V_NOSPAN    = 3

C ***** DEFINE RECORD-TYPE FATDEF (CF. IO USER'S TABLE 1-8)
        STRUCTURE /FATDEF/
                BYTE            FAT$B_RTYPE
                BYTE            FAT$B_RATTRIB
                INTEGER*2       FAT$W_RSIZE

                UNION
                        MAP
                                INTEGER*4       FAT$L_HIBLK
                        END MAP
                        MAP
                                INTEGER*2       FAT$W_HIBLKH
                                INTEGER*2       FAT$W_HIBLKL
                        END MAP
                END UNION
                UNION
                        MAP
                                INTEGER*4       FAT$L_EFBLK
                        END MAP
                        MAP
                                INTEGER*2       FAT$W_EFBLKH
                                INTEGER*2       FAT$W_EFBLKL
                        END MAP
                END UNION

                INTEGER*2       FAT$W_FFBYTE
                BYTE                FAT$B_BKTSIZE
                BYTE                FAT$B_VFCSIZE
                INTEGER*2       FAT$W_MAXREC
                INTEGER*2       FAT$W_DEFEXT

                BYTE                FAT$B_RESERVED(8)

                INTEGER*2       FAT$W_VERSIONS
        END STRUCTURE
```

```
       RECORD/FATDEF/FAT_BLOCK                       ! FAT BLOCK

        INCLUDE '($FIBDEF)'
      ! FILE INFORMATION BLOCK
        RECORD/FIBDEF/FI_BLOCK
        COMMON/FIB_DESCRIPTOR/FIB_LENGTH, FIB_ADDRESS
        INTEGER*4 FIB_LENGTH, FIB_ADDRESS

C ++ TRANSFER CURRENT FIB TO COMMON
        CALL LIB$MOVC3(22, I_FIB, FI_BLOCK)     ! SAVED FIB IS
                                                ! ALWAYS 22 BYTES

        FIB_LENGTH = 48                         ! FULL LENGTH
        FIB_ADDRESS = %LOC(FI_BLOCK)            ! LOCAL ADDRESS

C ++ SET UP FIB FOR WRITE ACCESS
        FI_BLOCK.FIB$L_ACCTL = 0                ! READ ONLY

C ++ SET UP FILE ATTRIBUTE LIST
        ATR_BLOCK.ATR$W_SIZE = 14               ! BYTES IN FAT BLOCK
        ATR_BLOCK.ATR$W_TYPE = ATR$C_RECATTR    ! SPECIFY FAT BLOCK
        ATR_BLOCK.ATR$L_ADDR = %LOC(FAT_BLOCK)  ! ADDRESS OF FAT BLOCK

C .. MOVE TO ATTRIBUTE CONTROL BLOCK
        CALL LIB$MOVC3(14, ATR_BLOCK, IACB)

C .. TERMINATE LIST
        LACB(2) = 0

C ++ ACCESS FILE
        L_STATUS = 0
        L_STATUS = SYS$QIOW(
        1       ,                               ! DEFAULT EFN = 0
        1       %VAL(I_CHANNEL),
        1       %VAL(IO$_ACCESS),
        1       IOSB,,,
        1       FIB_LENGTH,,,,
        1       IACB,)

        IER = 125
        IF(.NOT.L_STATUS) GOTO 910
        CALL LIB$MOVC3(2, IOSB(1), L_STATUS)
        IER = 130
        IF(.NOT.L_STATUS) GOTO 910

C ++ SET UP FAT BLOCK (CF. IO USER'S TABLE 1-8)
        FAT_BLOCK.FAT$L_EFBLK = L_BLOCKS + 1
        I_WORD = 0
        CALL LIB$MOVC3(2, FAT_BLOCK.FAT$W_EFBLKL, I_WORD)
        FAT_BLOCK.FAT$W_EFBLKL = FAT_BLOCK.FAT$W_EFBLKH
        CALL LIB$MOVC3(2, I_WORD, FAT_BLOCK.FAT$W_EFBLKH)
        FAT_BLOCK.FAT$W_FFBYTE = 0
```

```
C ++ SET FIB FOR TRUNCATE
        I_WORD = 0
        FI_BLOCK.FIB$W_EXCTL = I_WORD .OR. FIB$M_TRUNC
        FI_BLOCK.FIB$L_EXSZ = 0
        FI_BLOCK.FIB$L_EXVBN = L_BLOCKS + 1
        FIB_LENGTH = 44

C ++ DO TRUNCATE
        L_STATUS = 0
        L_STATUS = SYS$QIOW(
        1       ,                                 ! DEFAULT EFN = 0
        1       %VAL(I_CHANNEL),
        1       %VAL(IO$_MODIFY),
        1       IOSB,,,
        1       FIB_LENGTH,,,,
        1       IACB,)

        IER = 140
        IF(.NOT.L_STATUS) GOTO 910
        CALL LIB$MOVC3(2, IOSB(1), L_STATUS)
        IER = 145
        IF(.NOT.L_STATUS) GOTO 910

        IRES = 1
        RETURN

910     CONTINUE
        WRITE(6, 6910) IER, IRES, L_STATUS
6910    FORMAT(' QIO_TRUNCATE -- ERROR -- ', 2I7, Z10.1)
        IRES = -IER
        RETURN
        END
```

```
C ***** QIO_DELETE : DELETE FILE ON CLOSE
        SUBROUTINE QIO_DELETE(I_CHANNEL,        ! I/O CHANNEL
        1                     I_UNIT,           ! LOGICAL UNIT
        1                     I_FIB,            ! FIB
        1                     IRES)

        IMPLICIT INTEGER*4 (L, S)
        IMPLICIT BYTE (C)
        IMPLICIT REAL*8 (D)
        INCLUDE '($IODEF)'
        INTEGER*2 IOSB(4)
        COMMON/FIB_DESCRIPTOR/
        1  FIB_LENGTH,
        1  FIB_ADDRESS
        INTEGER*4 FIB_LENGTH, FIB_ADDRESS       ! FIB_LENGTH = 22 BYTES

C ++ ISSUE ACP QIO TO DELETE FILE
        FIB_LENGTH = 22
        FIB_ADDRESS = %LOC(I_FIB)
        L_FUNCTION = IO$_DELETE .OR. IO$M_DELETE
        L_STATUS = 0
        L_STATUS = SYS$QIOW(
        1  ,                                    ! DEFAULT EFN = 0
        1  %VAL(I_CHANNEL),                     ! I/O CHANNEL
        1  %VAL(L_FUNCTION),                    ! DELETE FUNCTION
        1  IOSB,                                ! I/O STATUS BLOCK
        1  ,,
        1  FIB_LENGTH,,,,,)                     ! FIB DESCRIPTOR

        IER = 721
        IF(.NOT.L_STATUS) GOTO 910
        IER = 722
        CALL LIB$MOVC3(2, IOSB(1), L_STATUS)
        IF(.NOT.L_STATUS) GOTO 910

C ++ MAKE DARN SURE THIS PUP IS GONE
        CALL QIO_CLOSE(I_CHANNEL, I_UNIT, IRES)
        IER = 750
        IF(IRES .LT. 1) GOTO 910

        IRES = 1
        RETURN

C ++ BAD NEWS
910     CONTINUE
        WRITE(6, 6910) IER, IRES, L_STATUS
6910    FORMAT(' QIO_DELETE -- ERROR -- ', 2I7, Z10.1)
        IRES = -IER
        RETURN
        END
```

```
C ***** QIO_CLOSE : CLOSE FILE OPENED FOR QIO USE

        SUBROUTINE QIO_CLOSE( I_CHAN,       ! I/O CHANNEL
        1                     I_UNIT,       ! FORTRAN LOGICAL UNIT
        1                     IRES)         ! RESULT

        INTEGER*4 L_STATUS, SYS$DASSGN

        L_STATUS = 0
        L_STATUS = SYS$DASSGN(%VAL(I_CHAN))

        IF(.NOT.L_STATUS) GOTO 910
        CLOSE(I_UNIT, ERR=800)

800     CONTINUE

        IRES = 1
        RETURN

910     CONTINUE
        WRITE(6, 6910) L_STATUS
6910    FORMAT(' QIO_CLOSE -- ERROR -- ', Z10.1)
        IRES = -1
        RETURN
        END
```

```
C ***** RMS_$OPEN : USEROPEN FORTRAN FUNCTION FOR RMS BLOCK I/O
C                   (FILE STATUS = 'OLD')

        INTEGER*4 FUNCTION RMS_$OPEN(FA_BLOCK, RA_BLOCK, LOGICAL_UNIT)

        IMPLICIT INTEGER*4 (L, S)

        INCLUDE '($FABDEF)'
        RECORD/FABDEF/FA_BLOCK

        INCLUDE '($RABDEF)'
        RECORD/RABDEF/RA_BLOCK

        INCLUDE '($SYSSRVNAM)'

C ++ SET BLOCK I/O BIT IN FAB
        FA_BLOCK.FAB$B_FAC = FA_BLOCK.FAB$B_FAC .OR. FAB$M_BIO
        FA_BLOCK.FAB$B_FAC = FA_BLOCK.FAB$B_FAC .OR. FAB$M_BRO

C ++ SET GET ACCESS (NEEDED FOR READONLY)
        FA_BLOCK.FAB$B_FAC = FA_BLOCK.FAB$B_FAC .OR. FAB$M_GET

C ++ SET RECORD LOCK FOR SHARED FILES IN FAB
        FA_BLOCK.FAB$B_SHR = FA_BLOCK.FAB$B_SHR .OR. FAB$M_UPI

C ++ REMOVE TRUNCATE OPTION
        FA_BLOCK.FAB$B_FAC = FA_BLOCK.FAB$B_FAC .AND. (.NOT.FAB$M_TRN)

C ++ OPEN
        STATUS = SYS$OPEN(FA_BLOCK)
        IER = 10
        IF(.NOT.STATUS) GOTO 910

C ++ SET BLOCK I/O BIT IN RAB
        RA_BLOCK.RAB$L_ROP = RA_BLOCK.RAB$L_ROP .OR. RAB$M_BIO

C ++ CONNECT RECORD STREAM
        STATUS = SYS$CONNECT(RA_BLOCK)
        IER = 20
        IF(.NOT.STATUS) GOTO 910

C ++ RETURN
        RMS_$OPEN = STATUS
        RETURN

C ++ ERROR MESSAGE
910     CONTINUE
        RMS_$OPEN = STATUS
        WRITE(6, 6910) IER, STATUS
6910    FORMAT(' RMS_$OPEN -- ERROR --', I7, Z12.12)
        RETURN
        END
```

```
C ***** RMS_$CREATE : USEROPEN FORTRAN FUNCTION FOR RMS BLOCK I/O
C                     (FILE STATUS = 'NEW' OR 'UNKNOWN')

        INTEGER*4 FUNCTION RMS_$CREATE(FA_BLOCK, RA_BLOCK, LOGICAL_UNIT)

        IMPLICIT INTEGER*4 (L, S)

        INCLUDE '($FABDEF)'
        RECORD/FABDEF/FA_BLOCK

        INCLUDE '($RABDEF)'
        RECORD/RABDEF/RA_BLOCK

        INCLUDE '($SYSSRVNAM)'

C ++ SET BLOCK I/O BIT IN FAB
        FA_BLOCK.FAB$B_FAC = FA_BLOCK.FAB$B_FAC .OR. FAB$M_BIO
        FA_BLOCK.FAB$B_FAC = FA_BLOCK.FAB$B_FAC .OR. FAB$M_BRO

C ++ SET GET ACCESS (NEEDED FOR READONLY)
        FA_BLOCK.FAB$B_FAC = FA_BLOCK.FAB$B_FAC .OR. FAB$M_GET

C ++ SET RECORD LOCK FOR SHARED FILES IN FAB
        FA_BLOCK.FAB$B_SHR = FA_BLOCK.FAB$B_SHR .OR. FAB$M_UPI

C ++ REMOVE TRUNCATE OPTION
        FA_BLOCK.FAB$B_FAC = FA_BLOCK.FAB$B_FAC .AND. (.NOT.FAB$M_TRN)

C ++ OPEN
        STATUS = SYS$CREATE(FA_BLOCK)
        IER = 10
        IF(.NOT.STATUS) GOTO 910

C ++ SET BLOCK I/O BIT IN RAB
        RA_BLOCK.RAB$L_ROP = RA_BLOCK.RAB$L_ROP .OR. RAB$M_BIO

C ++ CONNECT RECORD STREAM
        STATUS = SYS$CONNECT(RA_BLOCK)
        IER = 20
        IF(.NOT.STATUS) GOTO 910

C ++ RETURN
        RMS_$CREATE = STATUS
        RETURN

C ++ ERROR MESSAGE
910     CONTINUE
        RMS_$CREATE = STATUS
        WRITE(6, 6910) IER, STATUS
6910    FORMAT(' RMS_$CREATE -- ERROR --', I7, Z12.12)
        RETURN
        END
```

```
C ***** RMS_OPEN : OPEN FILE FOR RMS BLOCK I/O

        SUBROUTINE RMS_OPEN(  I_RW_FLAG,  ! 0=READ (OLD)
                                          ! 1=WRITE (OLD)
                                          ! 2=WRITE (NEW)
                                          ! 3=WRITE (UNKNOWN)
        1                     I_UNIT,     ! LOGICAL UNIT
        1                     C_FILE,     ! FILE NAME
        1                     L_BLOCKS,   ! INITIAL BLOCKS (NEW ONLY)
        1                     IRES)       ! RESULT

        PARAMETER W_P_B = 256                   ! WORDS PER DISK BLOCK
        EXTERNAL RMS_$OPEN                      ! C.F. Appendix C.
        EXTERNAL RMS_$CREATE                    ! DITTO.

        INTEGER*4 L_BLOCKS
        BYTE C_FILE(1)

C ++ DO OPEN (DEPENDS ON VALUE OF I_RW_FLAG, I_UNIT, C_FILE)
        IRES = 0
        I_SIZE = W_P_B / 2                ! REALS PER BLOCK
        IF(I_RW_FLAG .EQ. 0) THEN         ! READ OLD FILE
                IER = 10
                OPEN(
        1       UNIT=I_UNIT,
        1       FILE=C_FILE,
        1       READONLY,
        1       STATUS='OLD',
        1       USEROPEN=RMS_$OPEN,
        1       SHARED,
        1       ERR=910)
                IRES = I_RW_FLAG + 1      ! RESULT
        ELSE IF(I_RW_FLAG .EQ. 1) THEN    ! WRITE OLD FILE
                IER = 20
                OPEN(
        1       UNIT=I_UNIT,
        1       FILE=C_FILE,
        1       STATUS='OLD',
        1       ORGANIZATION='SEQUENTIAL',
        1       CARRIAGECONTROL='NONE',
        1       FORM='UNFORMATTED',
        1       ACCESS='SEQUENTIAL',
        1       RECORDTYPE='FIXED',
        1       RECORDSIZE=I_SIZE,
        1       USEROPEN=RMS_$OPEN,
        1       SHARED,
        1       ERR=910)
                IRES = I_RW_FLAG + 1      ! RESULT
```

```
      ELSE IF(I_RW_FLAG .EQ. 2) THEN     ! WRITE NEW FILE
              IER = 30
              OPEN(
      1       UNIT=I_UNIT,
      1       FILE=C_FILE,
      1       STATUS='NEW',
      1       INITIALSIZE=L_BLOCKS,
      1       ORGANIZATION='SEQUENTIAL',
      1       CARRIAGECONTROL='NONE',
      1       FORM='UNFORMATTED',
      1       ACCESS='SEQUENTIAL',
      1       RECORDTYPE='FIXED',
      1       RECORDSIZE=I_SIZE,
      1       USEROPEN=RMS_$CREATE,
      1       SHARED,
      1       ERR=910)
              IRES = I_RW_FLAG + 1       ! RESULT
      ELSE IF(I_RW_FLAG .EQ. 3) THEN     ! WRITE UNKNOWN FILE
              IER = 40
              OPEN(
      1       UNIT=I_UNIT,
      1       FILE=C_FILE,
      1       STATUS='UNKNOWN',
      1       INITIALSIZE=L_BLOCKS,
      1       ORGANIZATION='SEQUENTIAL',
      1       CARRIAGECONTROL='NONE',
      1       FORM='UNFORMATTED',
      1       ACCESS='SEQUENTIAL',
      1       RECORDTYPE='FIXED',
      1       RECORDSIZE=I_SIZE,
      1       USEROPEN=RMS_$CREATE,
      1       SHARED,
      1       ERR=910)
              IRES = I_RW_FLAG + 1       ! RESULT
      END IF

C ++ DONE
      RETURN                             ! SUCCESS >0

C ++ ERROR ON OPEN
910   CONTINUE
      WRITE(6, 6910) IER
6910  FORMAT(' RMS_OPEN -- ERROR -- ', I6)
      IRES = -IER
      RETURN
      END
```

```
C ***** RMS_GET : READ WITH RMS SERVICE $READ (CF. RMS MANUAL)
C NB : RETURNS A STATUS CODE OF 0 FOR EOF ... CHECK BYTE COUNTS

        SUBROUTINE RMS_GET(   I_UNIT,           ! FORTRAN LUN
        1                     I_WORDS,          ! WORDS TO READ
        1                     I_BUFFER,         ! DATA BUFFER
        1                     L_BASE,           ! BASE WORD IN FILE
        1                     IRES)             ! RESULT (WORDS READ)
                                                ! (OR ERROR STATUS)

        IMPLICIT INTEGER*4 (L, F, S)
        INTEGER*2 I_BUFFER(1)
        INTEGER*4 SYS$READ, FOR$RAB
        PARAMETER W_P_B = 256                   ! WORDS PER DISK BLOCK
        INTEGER*2 I_BLOCK(W_P_B)
        INCLUDE '($RMSDEF)'

        L_UNIT = I_UNIT
        L_RAB_ADDRESS = FOR$RAB(L_UNIT)         ! GET RAB ADDRESS

        LBLK1 = L_BASE/W_P_B + 1                ! FIRST BLOCK TO READ
        I_FIRST = (W_P_B*2)                     ! READ FIRST BLOCK

        I_MOVE = 0
        CALL LOAD_RAB(%VAL(L_RAB_ADDRESS),      ! LOAD RAB WITH READ
        1       I_BLOCK,                        ! PARAMETERS
        1       I_FIRST,
        1       LBLK1)

        STATUS = SYS$READ(%VAL(L_RAB_ADDRESS))  ! DO READ
        IF(STATUS .EQ. RMS$_EOF) THEN                        ! RETRIEVE
            CALL GET_READ_SIZE(%VAL(L_RAB_ADDRESS), I_LEN)   ! ACTUAL
                                                             ! BYTES READ
            STATUS = RMS$_SUC
            I_NEED = L_BASE - (LBLK1-1)*W_P_B
            I_NEED = I_NEED * 2
            I_FIRST = MAX(0, I_LEN-I_NEED)
            IF(I_FIRST .GT. 0) I_FIRST = I_LEN
            I_LEN = 0
        END IF
        IER = 10
        IF(.NOT.STATUS) GOTO 910

C ++ SHIFT DATA TO FIRST WORD OF BUFFER
        IF(I_FIRST .GT. 0) THEN
        ISHFT = L_BASE - (LBLK1-1)*W_P_B        ! OFFSET OF USABLE DATA
        I_MOVE = I_FIRST/2 - ISHFT
        I_MOVE = MIN(I_MOVE, I_WORDS)
        CALL LIB$MOVC3(I_MOVE*2, I_BLOCK(ISHFT+1), I_BUFFER(1))
        END IF

        IF(I_FIRST .LT. W_P_B*2) GOTO 800       ! DONE (EOF FOUND)

        I_GET = I_MOVE + 1
```

```
C ++ GET THE REST OF THE DATA
        I_LEN = (I_WORDS-I_MOVE) * 2            ! BYTES LEFT
        IF(I_LEN .LE. 0) GOTO 800               ! ALL DONE
        LBLK1 = LBLK1 + 1

        CALL LOAD_RAB(%VAL(L_RAB_ADDRESS),      ! LOAD RAB WITH READ
        1       I_BUFFER(I_GET),                ! PARAMETERS
        1       I_LEN,
        1       LBLK1)

        STATUS = SYS$READ(%VAL(L_RAB_ADDRESS))              ! DO READ
        IF(STATUS .EQ. RMS$_EOF) THEN
      ! RETRIEVE ACTUAL
            CALL GET_READ_SIZE(%VAL(L_RAB_ADDRESS), I_LEN)  ! BYTES READ
            STATUS = RMS$_SUC
        END IF
        IER = 20
        IF(.NOT.STATUS) GOTO 910

C ++ ALL DONE
800     CONTINUE
        IRES = I_MOVE + I_LEN/2
        RETURN

C ++ I HATE IT WHEN THAT HAPPENS
910     CONTINUE
        WRITE(6, 6910) IER, STATUS
6910    FORMAT(' RMS_GET -- ERROR -- ', I6, Z10.1)
        IRES = -IER
        RETURN
        END


C ***** GET_READ_SIZE : GET SIZE IN BYTES OF LAST READ
        SUBROUTINE GET_READ_SIZE(    RA_BLOCK,          ! RAB ADDRESS
        1                            I_LEN)             ! LENGTH IN BYTES

        IMPLICIT INTEGER*4 (L)
        INCLUDE '($RABDEF)'
        RECORD/RABDEF/RA_BLOCK

        I_LEN = 0
        CALL LIB$MOVC3(2, RA_BLOCK.RAB$W_RSZ, I_LEN)

        RETURN
        END
```

```
C ***** RMS_PUT : WRITE WITH RMS BLOCK I/O FUNCTION $WRITE

        SUBROUTINE RMS_PUT(    I_UNIT,      ! LUN
        1                      I_WORDS,     ! WORDS TO WRITE
        1                      I_BUFFER,    ! DATA BUFFER
        1                      L_BASE,      ! BASE WORD IN FILE
        1                      IRES)        ! RESULT (WORDS WRITTEN)
                                            ! OR ERROR STATUS

        IMPLICIT INTEGER*4 (L, S, F)
        INTEGER*2 I_BUFFER(1)
        PARAMETER W_P_B = 256               ! WORDS PER DISK BLOCK
        INTEGER*2 I_BLOCK(W_P_B)            ! ONE DISK BLOCK BUFFER

        LBLK1 = L_BASE/W_P_B + 1            ! FIRST BLOCK TO WRITE
        I_MOVE = 0                          ! WORDS MOVED TO I_BLOCK
        I_READ = W_P_B                      ! WORDS FOR 1 BLOCK
        L_UNIT = I_UNIT
        L_RAB_ADDRESS = FOR$RAB(L_UNIT)     ! GET RAB ADDRESS

        IF((L_BASE/W_P_B*W_P_B) .EQ. L_BASE) GOTO 50  ! WRITE STARTS ON BLOCK
                                                      ! BOUNDARY

        L_GO = L_BASE/W_P_B * W_P_B
        CALL RMS_GET(    I_UNIT,            ! FORTRAN LUN
        1                I_READ,            ! WORDS TO READ
        1                I_BLOCK,           ! DATA BUFFER
        1                L_GO,              ! BASE WORD IN FILE
        1                IRES)              ! RESULT (BYTES READ)
                                            ! (OR ERROR STATUS)

        IER = 10
        IF(IRES .LT. 0) GOTO 910

C ++ SHIFT DATA FROM INPUT BUFFER
        ISHFT = L_BASE - (LBLK1-1)*W_P_B    ! OFFSET OF USABLE DATA
        I_MOVE = W_P_B - ISHFT              ! WORDS TO MOVE FROM I_BUFFER
        I_MOVE = MIN(I_WORDS, I_MOVE)
        CALL LIB$MOVC3(I_MOVE*2, I_BUFFER, I_BLOCK(ISHFT+1))

C ++ REWRITE PARTIAL FIRST BLOCK

        CALL LOAD_RAB(%VAL(L_RAB_ADDRESS),! LOAD RAB WITH WRITE
        1       I_BLOCK,                    ! PARAMETERS
        1       I_READ*2,
        1       LBLK1)

        STATUS = SYS$WRITE(%VAL(L_RAB_ADDRESS))    ! DO WRITE
        IER = 20
        IF(.NOT.STATUS) GOTO 910
```

```
C ++ WRITE MAIN BUFFER (ALL BUT LAST BLOCK)
        LBLK1 = LBLK1 + 1                  ! STARTING BLOCK TO WRITE
                                           ! DON'T INCREMENT, IF YOU
                                           ! SKIPPED FIRST PARTIAL BLOCK.
50      CONTINUE
        J_WORDS = I_WORDS - I_MOVE         ! COUNT WORDS LEFT TO WRITE
        K_WORDS = (J_WORDS/W_P_B) * W_P_B  ! ALIGN ON BLOCK BOUNDARY
        IF(K_WORDS .EQ. 0) GOTO 70         ! 1ST = LAST = PARTIAL

        K_BYTES = K_WORDS * 2              ! BYTES FOR THIS WRITE
        CALL LOAD_RAB(%VAL(L_RAB_ADDRESS),! LOAD RAB WITH WRITE
        1       I_BUFFER(I_MOVE+1),        ! PARAMETERS
        1       K_BYTES,
        1       LBLK1)

        STATUS = SYS$WRITE(%VAL(L_RAB_ADDRESS))       ! DO WRITE
        IER = 50
        IF(.NOT.STATUS) GOTO 910

C ++ REWRITE PARTIAL LAST BLOCK IF NECESSARY
        J_WORDS = J_WORDS - K_WORDS        ! WORDS LEFT TO WRITE
70      CONTINUE                           ! GO HERE IMMEDIATELY, WHEN
                                           ! ONLY PARTIAL BLOCKS WRITTEN.
        IER = 70
        IF(J_WORDS .GT. W_P_B) GOTO 910    ! S/B <= 1 BLOCK
        IF(J_WORDS .EQ. 0) GOTO 800        ! ALL DONE ... SKIP TO END

C .. GET PARTIAL LAST BLOCK
        LBLK1 = (L_BASE+I_WORDS+W_P_B-1) / W_P_B  ! LAST BLOCK TO WRITE

        L_GO = (LBLK1-1) * W_P_B
        CALL RMS_GET(   I_UNIT,            ! FORTRAN LUN
        1               I_READ,            ! WORDS TO READ
        1               I_BLOCK,           ! DATA BUFFER
        1               L_GO,              ! BASE WORD IN FILE
        1               IRES)              ! RESULT (BYTES READ)
                                           ! (OR ERROR STATUS)
        IER = 80
        IF(IRES .LT. 0) GOTO 910

C ++ SHIFT DATA FROM INPUT BUFFER
        J_MOVE = I_MOVE + K_WORDS          ! WORDS ALREADY WRITTEN
        CALL LIB$MOVC3(J_WORDS*2, I_BUFFER(J_MOVE+1), I_BLOCK(1))

        CALL LOAD_RAB(%VAL(L_RAB_ADDRESS),! LOAD RAB WITH WRITE
        1       I_BLOCK,                   ! PARAMETERS
        1       I_READ*2,
        1       LBLK1)

        STATUS = SYS$WRITE(%VAL(L_RAB_ADDRESS))       ! DO WRITE
        IER = 90
        IF(.NOT.STATUS) GOTO 910
```

```
C ++ FEETS, DO YOUR STUFF
800     CONTINUE
        IRES = I_WORDS
        RETURN

C ++ I HATE IT WHEN THAT HAPPENS
910     CONTINUE
        WRITE(6, 6910) IER, STATUS
6910    FORMAT(' RMS_PUT -- ERROR -- ', I6, Z10.1)
        IRES = -IER
        RETURN
        END


C ***** LOAD_RAB : STUFF RAB WITH VALUES NEEDED FOR $READ AND $WRITE
        SUBROUTINE LOAD_RAB(  RA_BLOCK,           ! RAB ADDRESS
        1                     I_BUFFER,           ! USER BUFFER
        1                     I_LEN,              ! LENGTH IN BYTES
        1                     LBLK)               ! VBN

        IMPLICIT INTEGER*4 (L)
        INCLUDE '($RABDEF)'
        RECORD/RABDEF/RA_BLOCK

C ++ STUFF RAB
        RA_BLOCK.RAB$L_BKT = LBLK                 ! VBN
        RA_BLOCK.RAB$L_UBF = %LOC(I_BUFFER)       ! READ BUFFER ADDRESS
        RA_BLOCK.RAB$L_RBF = %LOC(I_BUFFER)       ! WRITE BUFFER ADDRESS
        CALL LIB$MOVC3(2, I_LEN, RA_BLOCK.RAB$W_USZ)   ! READ LENGTH
        CALL LIB$MOVC3(2, I_LEN, RA_BLOCK.RAB$W_RSZ)   ! WRITE LENGTH

C .. MAKE SURE TRUNCATE ON PUT BIT IS DISARMED
        RA_BLOCK.RAB$L_ROP = RA_BLOCK.RAB$L_ROP .AND. (.NOT.RAB$M_TPT)

        RETURN
        END
```