Y28-6604-1

Program Logic

IBM System/360 Operating System

Sequential Access Methods

Program Number 360S-DM-508

This publication describes the internal logic of the routines of the queued sequential access method, the basic sequential access method, and the basic partitioned access method of IBM System/360 Operating System. Program Logic Manuals are intended for use by IBM customer engineers involved in program maintenance, and by system programmers involved in altering the program design. Program logic information is not necessary for program operation and use; therefore, distribution of this manual is limited to persons with program maintenance or modification responsibilities.

PREFACE

This publication describes the sequential access method facilities in IBM Operating System/360. It describes routines in five categories:

- Queued sequential access method routines that cause storage and retrieval of data records arranged in sequential order.
- Basic sequential access method routines that cause storage and retrieval of data blocks arranged in sequential order.
- Basic partitioned access method routines that cause storage and retrieval of data blocks in a member of a partitioned data set, and construct entries and search for entries in the directory of a partitioned data set.
- Executors that operate with input/output support routines.
- Buffer pool management routines that furnish buffer space in main storage.

PREREQUISITE PUBLICATIONS

Knowledge of the information in the following publications is required for an understanding of this publication:

IBM System/360 Operating System: Data Management, Form C28-6537

IBM System/360 Operating System: Introduction to Control Program Logic, Program Logic Manual, Form Y28-6605

RECOMMENDED READING

The publication IBM System/360 Operating System: Control Program Services, Form C28-6541, provides useful information.

Second Edition (January 1967)

This publication is a major revision of Form Y28-6604-0 and obsoletes it. Significant additional material has been added; material released in a Technical Newsletter has been included, and prior material has been updated.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer using a special print chain.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.

A form for readers' comments appears at the back of this publication. It may be mailed directly to IBM. Address any additional comments concerning this publication to the IBM Corporation, Programming Systems Publications, Department D58, PO Box 390, Poughkeepsie, N. Y. 12602

ILLUSTRATIONS

FIGURES

TABLES

Sequential access methods are programming techniques for causing the storage and retrieval of data arranged in sequential order. Sequential access method facilities in Operating System/360 consist of routines in five categories:

- Queued sequential access method (QSAM) routines.
- Basic sequential access method (BSAM) routines.
- Basic partitioned access method (BPAM) routines.
- Sequential access method executors.
- Buffer pool management routines.

A processing program using QSAM routines deals with records. For input, QSAM routines turn the blocks of data of the channel programs into a stream of input records for the processing program; for output, QSAM routines collect the successive output records of the processing program into blocks of data to be written by channel programs.

A processing program using BSAM routines deals with blocks of data. For input, BSAM routines cause a channel program to read a block of data for the processing program; for output, BSAM routines cause a channel program to write a block of data for the processing program. BSAM routines are also used to read and write blocks of data for members of a partitioned data set.

A processing program using BPAM routines also deals with blocks of data. For output, BPAM routines construct and cause writing of entries in the directory; for input, BPAM routines cause searching for and read entries in the directory. To read and write the blocks of the members, a processing program uses the BSAM routines.

Sequential access method executors are modules that operate with the OPEN, CLOSE, and EOV routines of I/O support. When a data control block is opened, an executor constructs control blocks and loads the access method routines unless the resident access method (RAM) option is used. If the RAM option is used, the selected QSAM or BSAM routines are permanently resident. When the end of a data set or volume is reached, an executor processes the pending input/output blocks. The five types of executor are: OPEN executor, CLOSE executor, SYNAD/EOV executor, FEOV executor, and EOV/new volume executor.

Buffer pool management routines form buffers in main storage and return main storage space (for buffers no longer needed) to available status. A buffer pool management routine is entered when a GETPOOL, BUILD, GETBUF, FREEBUF, or FREEPOOL macro-instruction is encountered in a program.

The GETPOOL and BUILD routines both form a pool of buffers in main storage. However, the GETPOOL routine also obtains the main storage space for the buffer pool. Main storage space must be provided by the processing program when the BUILD routine is used.

The GETBUF and FREEBUF routines handle individual buffers. GETBUF obtains a buffer from a buffer pool and FREEBUF returns a buffer to a buffer pool.

The FREEPOOL routine returns the main storage space used for a buffer pool.

Figure 1 shows the relationship among sequential access method routines, other portions of the control program, and the processing program. Certain routines (e.g., end-of-block routines and appendages) are identical for all three sequential access methods. Other routines (e.g., GET or PUT for QSAM and READ or WRITE for BSAM and BPAM) depend upon the access method used. (QSAM and BSAM also include control routines not shown in Figure 1.)

A processing program passes control to sequential access method routines via a macro-instruction. A GET, RELSE, PUT, PUTX, or TRUNC macro-instruction is used for QSAM, and a READ or WRITE macro-instruction is used for BSAM. The GET, PUT, READ, and WRITE routines pass control to the same end-of-block routines. However, a GET or a PUT routine passes control only when an end-of-block condition occurs, and a READ or a WRITE routine always passes control. An end-of-block routine causes the I/O supervisor to schedule a channel program for execution. The end-of-block routine then returns control to the GET or PUT routine (for QSAM) or to the READ or WRITE routine (for BSAM and BPAM).

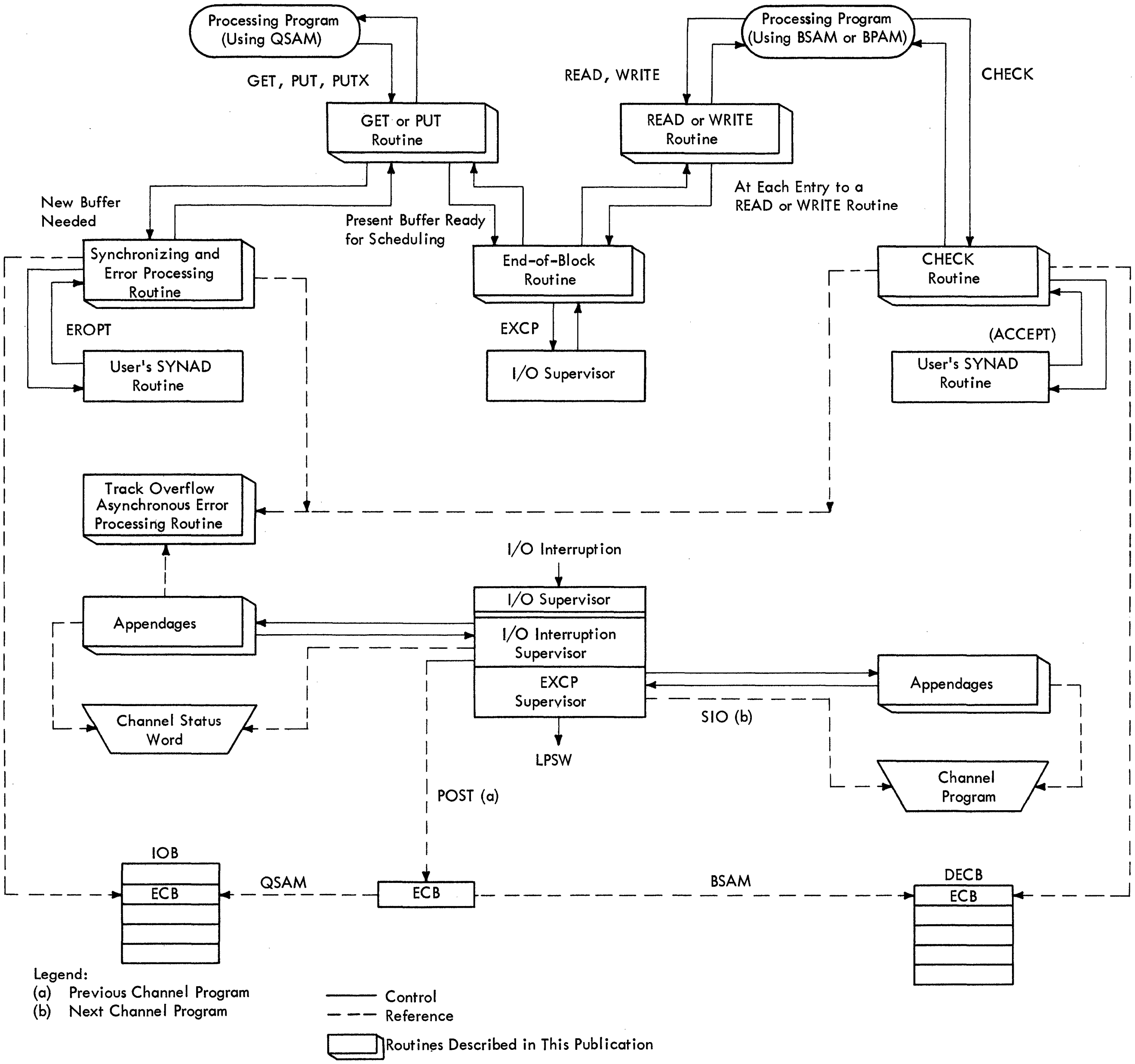


Figure 1. Flow of Control in QSAM, BSAM, and in BPAM for Members

After receiving control back from an end-of-block routine, a GET or a PUT routine passes control to a synchronizing and error processing routine. This routine examines the IOB to determine the status of the channel program. If the channel program is not yet executed, the synchronizing routine awaits execution. If the channel program executed successfully, control returns to the GET or PUT routine which returns control to the processing program. If execution of the channel program resulted in permanent errors the synchronizing routine causes control to pass to the user's SYNAD routine.

The asynchronous error processing routine gains control as a result of being scheduled by an appendage. The routine processes permanent error conditions that are encountered by a channel program for input data with track overflow record format. The routine establishes the address of the segment beyond the one in error.

After receiving control back from an end-of-block routine, the READ or WRITE routine returns control to the processing program. To determine the status of the channel program the processing program must pass control to a CHECK routine via a CHECK macro-instruction. A CHECK routine determines the status of the channel program by referring to the DECB. If the channel program is not yet executed, the CHECK routine awaits execution. If the channel program has been executed successfully, control returns to the processing program. If execution of the channel program resulted in a permanent error, a CHECK routine causes control to pass to the user's SYNAD routine.

When an I/O interruption occurs, the I/O interruption supervisor posts the status of the execution of the channel program in the event control block (ECB). For QSAM, the ECB is located in the input/output block (IOB); for BSAM, the ECB is located in the data event control block (DECB). The EXCP supervisor then receives control and causes the next scheduled channel program to be executed. Both the I/O interruption supervisor and the EXCP supervisor may use access method appendages.

QUEUED SEQUENTIAL ACCESS METHOD ROUTINES

Queued sequential access method (QSAM) routines cause storage and retrieval of records and furnish buffering and blocking facilities. There are six types of QSAM routines:

- GET routines.
- PUT routines.
- End-of-block routines.
- Synchronizing and error processing routines (including the track overflow asynchronous error processing routine).
- Appendages.
- Control routines.

Table 1 and Figure 2 show the relationship of QSAM routines, other portions of the operating system, and the processing program.

A GET or a PUT routine receives control after a GET, PUT, PUTX, RELSE, or TRUNC macro-instruction is encountered in a processing program. A GET routine presents an input record to the processing program and returns control to the processing program unless the input buffer is empty. A PUT routine accepts output records from the processing program and returns control to the processing program unless the output buffer is full.

When an input buffer is empty, or an output buffer is full, an end-of-block routine receives control from the GET or the PUT routine. An end-of-block routine provides device oriented data for the channel program. If normal channel-program scheduling is used, the routine passes control to the I/O supervisor (via an EXCP macro-instruction) to cause scheduling of the buffer. If chained channel-program scheduling is used, it attempts to add the present channel program to the last one in the chain of scheduled channel programs. If it is successful, control returns to the processing program. If it is unsuccessful, control passes to the I/O supervisor (via an EXCP macro-instruction).

After the end-of-block routine returns control, the GET or PUT routine passes control to a synchronizing and error processing routine. The synchronizing routine examines the next IOB to determine the status of the channel program. (For a description and diagram of the relationship of QSAM Control Blocks refer to Appendix B.)

Table 1. Flow of Control of QSAM Routines

Routine Passing Control	Condition	Routine Receiving Control
Processing Program	GET or PUT Macro-instruction	GET or PUT
GET or PUT	Buffer ready for scheduling	End-of-block
End-of-block	EXCP Macro-instruction	I/O Supervisor
I/O Supervisor	End	End-of-block
End-of-block	End	GET or PUT
GET or PUT	New buffer needed	Synchronizing
Synchronizing	Channel program executed without error	GET or PUT
GET or PUT	No other	Processing Program
Supervisor	I/O interruption	I/O Supervisor
I/O Supervisor	Appendage exit condition	Appendage
Appendage	End	I/O Supervisor
I/O Supervisor	End	Supervisor

Depending on the status of the execution, a synchronizing routine may retain control (using the WAIT macro-instruction), return control to the GET or PUT routine, or pass control to the user's SYNAD routine or to the SYNAD/EOV executor. (For a description of the SYNAD/EOV executor (IGC0005E), and the flow of control to and from it, refer to the section: "Sequential Access Method Executors.")

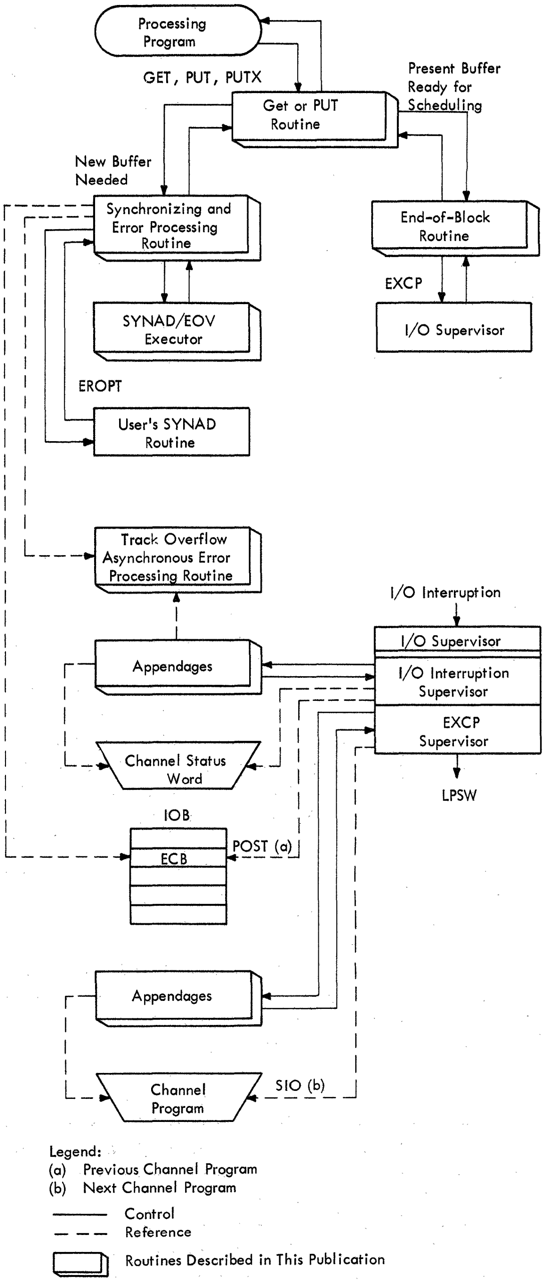


Figure 2. Flow of Control in QSAM

The track overflow asynchronous error processing routine gains control as a result of being scheduled by an appendage. The routine processes permanent error conditions that are encountered by a channel program for input data with track overflow record format. The routine establishes the address of the segment beyond the one in error.

Appendages receive control from the I/O supervisor and return control to the I/O supervisor. Some appendages operate with the I/O interruption supervisor, others operate with the EXCP supervisor.

Control routines (not shown in Figure 2) receive control from the processing program via the control macro-instructions (CNTRL, PRTOV). These QSAM routines control the printer and the card reader.

Appendix A contains decision tables that show, for each type of routine, the processing characteristics that differentiate the routines within that type.

GET ROUTINES

There are 14 different GET routines. A particular GET routine is used with a specific data set on the basis of the access condition options specified by the processing program for access to that data set.

A GET routine gains CPU control when a GET or a RELSE macro-instruction is encountered. The GET routine returns control to the processing program, unless either an input buffer is empty and ready to be scheduled for refilling or a new full input buffer is needed. If a buffer is ready to be scheduled for refilling, the GET routine passes control to an end-of-block routine. If a new full input buffer is needed, the GET routine passes control to a synchronizing and error processing routine. A GET routine presents the processing program with a record from a block of data in an input buffer filled by a channel program. A RELSE routine causes the present buffer to be considered empty and ready for refilling.

Every GET routine determines in each pass through the routine:

- The address of the next record.
- Whether an input buffer is empty and ready to be scheduled for refilling.
- Whether a new full input buffer is needed.

In each entry into a GET routine, the processing program is presented with the next record.

GET routines differ mainly in the buffering techniques they support. GET routines for simple buffering deal with buffers that are permanently associated with one DCB. GET routines for exchange buffering deal with buffers that are exchanged between the input DCB, the output DCB, and the processing program. The GET routine for the Update mode of OPEN uses simple buffering; it differs from other simple buffering GET routines in that the same buffer is used for both input and output. The manner in which a GET routine performs its processing depends on the buffering mode.

Simple buffering GET routines determine the address of the next record by referring to the DCB. To determine whether a buffer is empty and whether a new buffer is needed, these routines compare the beginning and ending address of the buffer. To present a record to the processing program, a simple buffering GET routine either moves the record to a processing program work area or permits processing to be performed in the buffer space. In the latter case, if the record is to become part of an output data set it must be moved to an output buffer.

Exchange buffering GET routines determine the address of the next record by referring to the channel program. To determine whether a buffer is empty and whether a new buffer is needed, these routines compare the beginning and ending address of the channel program. To present a record to the processing program, an exchange buffering GET routine presents the processing program with the buffer or buffer segment. The buffer (or segment) is exchanged with a work area of the processing program, or with a buffer (or segment) from an output DCB (by a PUT routine using exchange buffering).

The update mode GET routine determines the address of the next input record by referring to the DCB. (The next output record is the last input record.) To determine whether a new input buffer is needed, and whether the buffer is to be emptied (that is, whether the last block is to be updated) before being filled with a new block, the routine also refers to the DCB. The record is presented to the processing program, and accepted for updating, in the same buffer space.

The GET routine descriptions that follow are accordingly grouped as:

- Simple Buffering GET Routines.

- Exchange Buffering GET Routines.

- Update Mode GET Routine.

SIMPLE BUFFERING GET ROUTINES

Simple buffering GET routines use buffers whose beginning and ending addresses are in the data control block (DCB). The beginning address is in the field DCBRECAD (address of the next record); the ending address is in the field DCBEOBAD (address of the end of the buffer). In each pass through a routine, it determines:

- The address of the next record.

- Whether an input buffer is empty and ready to be scheduled for refilling.

- Whether a new full input buffer is needed.

If the records are unblocked, the address of the next record is always that of the next buffer.

If the records are blocked, a GET routine determines the address of the next record by adding the length of the last record to the address of the last record. The address of the last record is in the DCBRECAD field of the data control block (DCB). If the records are fixed-length blocked records the length of each record is in the DCBLRECL field. If the records are variable-length blocked records, the length of each record is in the length field of the record itself.

A GET routine determines whether a buffer is empty and ready for refilling, and whether a new full buffer is needed, by testing for an end-of-block (EOB) condition.

When a buffer is empty, a GET routine passes control to an end-of-block routine to refill the buffer. The buffers are filled for the first time by OPEN executor IGG01911. Thus the buffers are primed for the first entry into a GET routine.

When a new full buffer is needed, a GET routine obtains it by passing control to the Input Synchronizing and Error Processing routine (module IGG019AQ). The synchronizing routine updates the DCBIOBA field (thus pointing to the new buffer) and returns control to the GET routine. A GET routine updates the DCBRECAD field by inserting in it the starting address of the buffer from the channel program associated with the new IOB. To update the DCBEOBAD field a GET routine adds the actual length of the block read to the buffer starting address. These two fields, DCBRECAD and DCBEOBAD, define the available buffer.

For unblocked records, an EOB condition exists after every entry into the GET routine. For blocked records, an EOB condition exists when the values in the DCBRECAD field and the DCBEOBAD field are equal. In the move operating mode, the buffer can be scheduled for refilling as soon as the last record is moved out; accordingly, an EOB test is made after moving each record, to schedule the buffer as soon as possible. Another EOB test is made on the next entry to the routine to determine whether a new full buffer is needed. In the locate mode, the empty buffer is scheduled when the routine is entered, if the last record was presented in the preceding entry; accordingly, an EOB test is made on entry into the routine to determine both whether a buffer is empty and ready for refilling and also whether a new full buffer is needed.

When the processing program determines that the balance of the present buffer is to be ignored and the first record of the next buffer is desired, the processing program issues a RELSE macro-instruction. Control passes to a RELSE routine which sets an EOB condition.

The OPEN executor primes (that is, schedules for filling) the buffers if QSAM is used with a DCB opened for Input, Update, or Readback. (For the locate mode, all buffers except one are primed; for the move mode all buffers are primed.) The OPEN executor also sets an end-of-block condition; the first time that a GET routine gains control, it processes this condition in the way it normally does.

There are nine simple buffering GET routines. Table 2 lists the routines available and the conditions that cause a particular routine to be used. The OPEN executor selects one of the routines, loads it, and puts its address into the DCBGET field. The table shows, for example, that when the OPEN parameter list specifies Input and the DCB specifies the GET macro-instruction, simple buffering, the locate mode, and the fixed-length record format, routine IGG019AA is selected and loaded.

GET Module IGG019AA

Module IGG019AA presents the processing program with the address of the next fixed-length or undefined-length record. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Input

and the DCB specifies:

- GET
- Simple buffering
- Locate operating mode
- Fixed-length (unblocked, blocked, or blocked standard) or undefined-length record format.

The module consists of a GET routine and a RELSE routine.

The GET routine operates as follows:

- It receives control when a GET macro-instruction is encountered in a processing program.
- It tests for an EOB condition to determine whether a buffer is empty and ready for refilling and also whether a new buffer is needed. (When the OPEN executor primes the buffers, it schedules all buffers except one and sets an EOB condition.)
- If no EOB condition exists, it determines the address of the next record, and then presents the address to the processing program and returns control to the processing program.
- If an EOB condition exists, it issues a BALR instruction to pass the present buffer to the end-of-block routine to be scheduled for refilling. The GET routine issues another BALR instruction to obtain a new full buffer through the Input Synchronizing and Error Processing routine (module IGG019AQ). The GET routine then presents the address of the first record of the new buffer to the processing program and returns control to the processing program.

The RELSE routine causes an EOB condition by setting the DCBRECAD and DCBEOBAD fields so that they are equal; it then returns control to the processing program.

Table 2. Module Selector - Simple Buffering GET Modules

Access Conditions	Selections													
INPUT, GET, Simple Buffering	X	X	X	X	X	X	X	X	X					X
RDBACK, GET, Simple Buffering										X	X	X	X	
Locate operating mode	X	X	X							X	X			
Move operating mode				X	X	X	X	X	X			X	X	X
Fixed-length record format	X			X			X			X		X		
Undefined-length record format		X			X			X			X		X	
Variable-length record format			X			X			X					
Card reader, only a single buffer, CNTRL							X	X	X					
Character conversion for paper tape														X
GET Modules														
IGG019AA	X	X												
IGG019AB			X											
IGG019AC				X	X									
IGG019AD						X								
IGG019AG							X	X						
IGG019AH									X					
IGG019AM										X	X			
IGG019AN												X	X	
IGG019AT[1]														X

[1] This module also includes the Paper Tape Character Conversion Synchronizing and Error Processing routine.

GET Module IGG019AB

Module IGG019AB presents the processing program with the address of the next variable-length record. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Input

and the DCB specifies:

- GET
- Simple buffering
- Locate operating mode
- Variable-length (unblocked or blocked) record format.

The module consists of a GET routine and a RELSE routine.

The GET routine operates as follows:

- It receives control when a GET macro-instruction is encountered in a processing program.

- It determines the address of the next record and tests for an EOB condition to determine whether a buffer is empty and ready for refilling and also whether a new buffer is needed. (When the OPEN executor primes the buffers, it schedules all buffers except one and sets an EOB condition.)

- If no EOB condition exists, it presents the address of the next record to the processing program and returns control to the processing program.

- If an EOB condition exists, it issues a BALR instruction to pass the present buffer to the end-of-block routine to be scheduled for refilling. The GET routine issues another BALR instruction to obtain a new buffer through the Input Synchronizing and Error Processing routine (module IGG019AQ). The GET routine then presents the address of the first record of the new buffer to the processing program and returns control to the processing program.

The RELSE routine causes an EOB condition by setting the DCBRECAD and DCBEOBAD fields so that they are equal; it then returns control to the processing program.

GET Module IGG019AC

Module IGG019AC moves the next fixed-length or undefined-length record to the work area. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Input

and the DCB specifies:

- GET
- Simple buffering
- Move operating mode
- Fixed-length (unblocked, blocked, or blocked standard) or undefined-length record format

(but not the CNTRL macro-instruction). The module consists of a GET routine and a RELSE routine.

The GET routine operates as follows:

- It receives control when a GET macro-instruction is encountered in a processing program.

- It tests for an EOB condition to determine whether a new full buffer is needed. (When the OPEN executor primes the buffers, it sets this EOB condition for the first GET macro-instruction.)

- If no EOB condition exists, the routine moves the next record to the work area.

- If an EOB condition exists, the routine issues a BALR instruction to obtain a new buffer through the Input Synchronizing and Error Processing routine (module IGG019AQ), and moves the first record of the new buffer to the work area.

- It tests for a new EOB condition to determine whether a buffer is empty and ready for refilling. (For unblocked records, this condition exists at every entry into the routine.)

- If no new EOB condition exists, the routine returns control to the processing program.

- If a new EOB condition exists, the routine issues a BALR instruction to pass the present buffer to the end-of-block routine to be scheduled for refilling, and then returns control to the processing program.

The RELSE routine sets a bit in the DCB so that the GET routine passes the buffer for refilling and obtains a new full buffer the next time the routine is entered.

GET Module IGG019AD

Module IGG019AD moves the next variable-length length record to the work area. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Input

and the DCB specifies:

- GET
- Simple buffering
- Move operating mode
- Variable-length (unblocked or blocked) record format

(but not the CNTRL macro-instruction). The module consists of a GET and a RELSE routine.

The GET routine operates as follows:

- It receives control when a GET macro-instruction is encountered in a processing program.

- It tests for an EOB condition to determine whether a new full buffer is needed. (When the OPEN executor primes the buffers, it also sets an end-of-block condition for the first GET macro-instruction.)

- If an EOB condition exists, the routine issues a BALR instruction to obtain a new buffer through the Input Synchronizing and Error Processing routine (module IGG019AQ), and moves the first record to the work area.

- If no EOB condition exists, the routine moves the next record to the work area.

- It tests for a new EOB condition to determine whether a buffer is empty and ready for refilling. (For unblocked records, the condition exists after every entry to this routine.)

- If no new EOB condition exists, the routine returns control to the processing program.

- If a new EOB condition exists, the routine issues a BALR instruction to pass the present buffer to the end-of-block routine to be scheduled for refilling, and returns control to the processing program.

The RELSE routine sets a bit in the DCB so that the GET routine passes the buffer for refilling and obtains a new full buffer the next time the routine is entered.

GET Module IGG019AG (CNTRL - Card Reader)

Module IGG019AG moves the next fixed-length or undefined-length record to the work area without scheduling the buffer for refilling. To refill the buffer, the processing program issues a CNTRL macro-instruction. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Input

and the DCB specifies:

- GET
- Simple buffering
- Move operating mode
- Fixed-length (unblocked, blocked, or blocked standard) or undefined-length record format
- CNTRL (card reader).

The module consists of a GET routine and a RELSE routine.

The GET routine operates as follows:

- It receives control when a GET macro-instruction is encountered in a processing program.

- If an EOB condition exists, it resets the DCBRECAD and DCBEOBAD fields for the new buffer, and then tests for blocked records.

- If no EOB condition exists, it tests immediately for blocked records.

- For blocked records, it updates the DCBRECAD field, moves the present record to the work area, and returns control to the processing program.

- For unblocked records, it sets the DCBRECAD and DCBEOBAD fields so that they are equal, moves the present record to the work area, and returns control to the processing program.

The RELSE routine sets the value of the DCBEOBAD field equal to that of the DCBRECAD field to establish an EOB condition. Control then returns to the processing program.

GET Module IGG019AH (CNTRL - Card Reader)

Module IGG019AH moves the next variable-length record to the work area without scheduling the buffer for refilling. To refill the buffer, the processing program issues a CNTRL macro-instruction. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Input

and the DCB specifies:

- GET
- Simple buffering
- Move operating mode
- Variable-length (unblocked or blocked) record format
- CNTRL (card reader).

The module consists of a GET routine and a RELSE routine.

The GET routine operates as follows:

- It receives control when a GET macro-instruction is encountered in a processing program.

- If an EOB condition exists, it resets the DCBRECAD and DCBEOBAD fields for the new buffer, and then tests for blocked records.

- If no EOB condition exists, it tests immediately for blocked records.

- For blocked records, it updates the DCBRFCAD field, moves the present record to the work area, and returns control to the processing program.

- For unblocked records, it sets the DCBRECAD and DCBEOBAD fields so that they are equal, moves the present record to the work area, and returns control to the processing program.

The RELSE routine sets the value of the DCBEOBAD field equal to that of the DCBRECAD field to establish an EOB condition. Control then returns to the processing program.

GET Module IGG019AM (RDBACK)

Module IGG019AM presents the processing program with the address of the next record when the data set is opened for backward reading. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- RDBACK

and the DCB specifies:

- GET
- Simple buffering
- Locate operating mode
- Fixed-length (unblocked, blocked, or blocked standard) or undefined-length record format.

The module consists of a GET routine and a RELSE routine.

The GET routine operates as follows:

- It receives control when a GET macro-instruction is encountered in a processing program.
- It tests for an EOB condition.
- If no EOB condition exists, it determines the address of the next record by subtracting the DCBLRECL value from the DCBRECAD value. The routine presents the result to the processing program, and returns control to the processing program.
- If an EOB condition exists, it issues a BALR instruction to pass the present buffer to the end-of-block routine. The GET routine issues another BALR instruction to obtain a new buffer through the Input Synchronizing and Error Processing routine (module IGG019AQ). The GET routine then presents the address of the last record of the new buffer to the processing program, and returns control to the processing program.

The RELSE routine causes an EOB condition by setting the DCBRECAD and DCBEOBAD fields so that they are equal; it then returns control to the processing program.

Figure 3 illustrates the ordering of records using this module. When reading backwards under QSAM, each block is read from the tape from the end of the block to the beginning, each buffer is filled from the end of the buffer to the beginning, and the records are presented to the processing program in order of the record in the last segment of the buffer first, and the record in the first one last. In this manner of reading, buffering, and presenting, each record follows in backward sequence, from the record presented last out of one buffer to the record presented first out of the next buffer.

GET Module IGG019AN (RDBACK)

Module IGG019AN moves the next record to the work area when the data set is opened for backward reading. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- RDBACK

and the DCB specifies:

- GET
- Simple buffering
- Move operating mode
- Fixed-length (unblocked, blocked, or blocked standard) or undefined-length record format.

The module consists of a GET routine and a RELSE routine.

The GET routine operates as follows:

- It receives control when a GET macro-instruction is encountered in a processing program.
- It tests for an EOB condition.
- If no EOB condition exists, it moves the next record to the work area, and updates the DCBRECAD field by reducing it by the value of the DCBLRECL field.
- If an EOB condition exists, it issues a BALR instruction to obtain a new buffer through the Input Synchronizing and Error Processing routine (module IGG019AQ). The GET routine then moves the last record of the new buffer to the work area.
- It tests for a new EOB condition.

- If no new EOB condition exists, it returns control to the processing program.

- If a new EOB condition exists, it issues a BALR instruction to pass the present buffer to the end-of-block routine, and then returns control to the processing program.

The RELSE routine issues a BALR instruction to pass the present buffer to the end-of-block routine, and then returns control to the processing program.

Figure 3, described for GET module IGG019AM, also illustrates the ordering of records using this module.

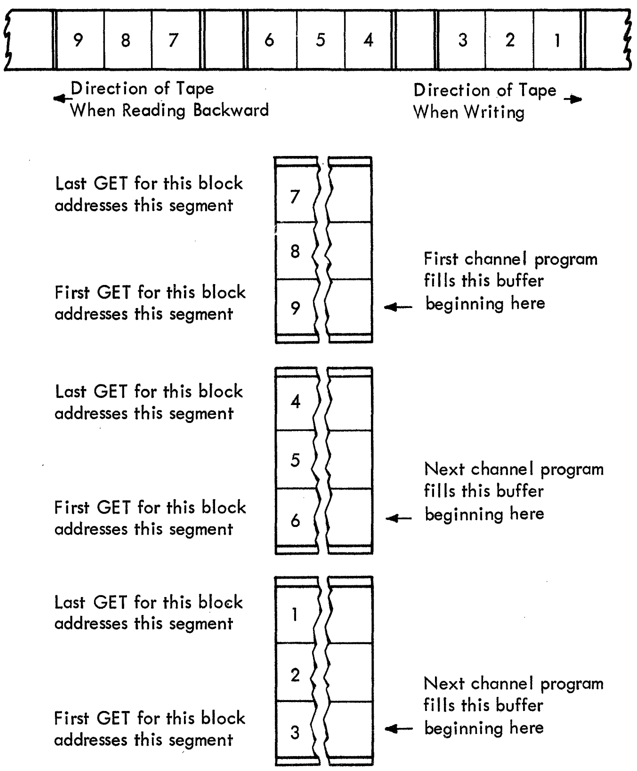

9
8
7
6
5
4
3
2
1
Direction of Tape When Reading Backward
Direction of Tape When Writing
Last GET for this block addresses this segment
7
8
First channel program fills this buffer beginning here
First GET for this block addresses this segment
9
Last GET for this block addresses this segment
4
5
Next channel program fills this buffer beginning here
First GET for this block addresses this segment
6
Last GET for this block addresses this segment
1
2
Next channel program fills this buffer beginning here
First GET for this block addresses this segment
3

Figure 3. Order of Records Using GET Routines for Data Sets Opened for RDBACK (IGG019AM,IGG019AN)

GET Module IGG019AT (Paper Tape Character Conversion)

Module IGG019AT converts paper tape characters into EBCDIC characters and moves them to the work area. The OPEN executor selects and loads this module (and one of the code conversion modules listed in Appendix D) if the OPEN parameter list specifies:

- Input

and the DCB specifies:

- GET
- Simple buffering
- Move operating mode
- Paper tape character conversion.

The module consists of a GET routine and a Paper Tape Character Conversion Synchronizing and Error Processing routine.

The GET routine operates as follows:

- It receives control when a GET macro-instruction is encountered in a processing program.

- It converts the next character and moves it to the work area.

- It continues converting and moving until one of the following conditions is met, with the stated effect:

  The number of characters specified in the DCBBLKSI field of the DCB have been moved: The routine returns control to the processing program.

  An EOB condition is encountered: The routine passes control to the end-of-block routine to refill the buffer, and then enters the Paper Tape Character Conversion Synchronizing and Error Processing routine to obtain a new buffer.

  An end-of-record character is encountered .(undefined-length records only): The routine returns control to the processing program.

  The tape is exhausted: The routine returns control to the processing program.

  A paper tape reader-detected error character is encountered: The routine moves the character to the work area without conversion and enters the Paper Tape Character Conversion Synchronizing and Error Processing routine.

- If one of the characters in the buffer is an undefined character, the module converts it to the hexadecimal character FF, moves it to the work area, and continues conversion. When one of the afore-mentioned conditions is met, control passes to the Paper Tape Character Conversion Synchronizing and Error Processing routine.

The Paper Tape Character Conversion Synchronizing and Error Processing routine operates as follows:

- For an EOB condition, the routine finds the next buffer, and returns control to the GET routine to resume converting and moving.

- For a reader-detected error character and for an undefined character, the routine passes control to the processing program's SYNAD routine. When control returns from the SYNAD routine, or if there is no SYNAD routine present, one of the error options is implemented.

- For the ACCEPT error option, the routine returns control to the processing program.

- For the SKIP error option, the routine fills the work area again.

- For the TERMINATE error option, or if no error option is specified, the routine issues the ABEND macro-instruction.

Appendix D lists the modules composed of the tables used for code conversion.

EXCHANGE BUFFERING GET ROUTINES

Exchange buffering GET routines use buffers whose addresses and lengths are stated in the channel program. For unblocked records, the buffer address and length are in one channel command word (CCW). For blocked records, the addresses of the buffer segments are in successive CCWs (though the segments themselves are not necessarily located next to one another). In each pass through an exchange buffering GET routine, it determines:

- The address of the next record.

- Whether an input buffer is empty and ready to be scheduled for refilling.

- Whether a new full input buffer is needed.

If the records are unblocked, a GET routine finds the address of the next record in the Read CCW for the next input buffer.

If the records are blocked, a GET routine finds the address of the next record in the next Read CCW for the same buffer. The next CCW is found by adding 8 to the address of the previously current CCW (the value stated in the DCBCCCW field in the DCB).

If an input buffer is empty and ready to be scheduled for refilling, a GET routine passes control to an end-of-block routine. The end-of-block routine passes control to the I/O supervisor to have it schedule the buffer. After scheduling, the I/O supervisor returns control to the end-of-block routine, and it returns control to the GET routine.

If a new full buffer is needed, a GET routine passes control to a synchronizing and error processing routine. The synchronizing routine enters the address of the input/output block (IOB) that points to that channel program into the DCBIOBA field in the DCB.

If an end-of-block condition exists then either an input buffer is empty and ready to be scheduled for refilling, or a new buffer is needed. An end-of-block condition exists for unblocked records during each pass through a routine; for blocked records it exists if the values in the fields DCBCCCW (the address of the current CCW) and DCBLCCW (the address of the last CCW) are equal.

In the locate operating mode, the empty buffer is scheduled when the routine is entered if the last record was presented in the preceding entry; accordingly a test for an end-of-block condition is made on entry to the routine to determine both whether a buffer is empty and also whether a new buffer is needed.

In the substitute operating mode, the buffer can be scheduled for refilling as soon as a work area has been substituted for the last buffer segment; accordingly, an end-of-block test is made before leaving the routine to determine whether the buffer is empty, and another end-of-block test is made on entry to the routine to determine whether a new buffer is needed.

A RELSE routine sets an end-of-block condition. This end-of-block condition is processed so that, when the GET routine is entered next, it operates as usual.

The OPEN executor primes (that is, schedules for filling) the buffers if QSAM is used with a DCB opened for Input. (For the locate mode, all buffers except one are primed; for the substitute mode all buffers are primed.) The OPEN executor also sets an end-of-block condition; the first time that a GET routine gains control, it processes this condition in the way it normally does.

There are four exchange buffering GET routines. Table 3 lists the routines available and the conditions that cause a particular routine to be used. The OPEN executor selects one of the routines, loads it, and places its address into the DCBGET field. The table shows, for example, that if Input, GET, exchange buffering, locate mode, and fixed-length blocked record format are specified module IGG019EA is selected for use.

Table 3. Module Selector - Exchange Buffering GET Modules

Access Conditions	Selections						
Input, GET, Exchange	X	X	X	X	X	X	X
Locate	X	X	X	X			
Substitute					X	X	X
Fixed-length	X	X			X		X
Variable-length			X				
Undefined-length				X		X	
Unblocked		X	X	X	X	X	
Blocked	X						X
GET Modules							
IGG019EA	X						
IGG019EB		X	X	X			
IGG019EC					X	X	
IGG019ED							X

GET Module IGG019EA

Module IGG019EA uses the locate mode to present the processing program with the address of the next fixed-length blocked record. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Input

and the DCB specifies:

- GET
- Exchange buffering
- Locate operating mode
- Fixed-length blocked record format.

The module consists of a GET routine and a RELSE routine.

The GET routine operates as follows:

- It receives control when a GET macro-instruction is encountered in a processing program.
- It tests for an end-of-block condition to determine whether a buffer is empty and ready for refilling and also whether a new full buffer is needed. (When the OPEN executor primes the buffers, it schedules all buffers except one and sets an end-of-block condition.)
- If no end-of-block condition exists, it presents the address of the next record (found in the next CCW), and returns control, to the processing program.
- If an end-of-block condition exists, the routine passes control to the end-of-block routine to cause scheduling of the buffer for refilling. On return of control, the GET routine passes control to the Input Synchronizing and Error Processing routine (module IGG019AQ) to obtain a new full buffer. On return of control, the GET routine then presents the address of the first record, and returns control, to the processing program.

The RELSE routine causes an end-of-block condition by setting the DCBCCCW and DCBLCCW fields equal and returns control to the processing program.

Note: If an input DCB using this module is paired with an output DCB using module IGG019EF (Output, PUT, Exchange), a PUTX macro-instruction addressed to the output DCB causes an exchange of the addresses of the current buffer segments of each DCB. These are found in the CCWs pointed to by the input and output DCBs.

GET Module IGG019EB

Module IGG019EB uses the locate mode to present the processing program with the address of the next unblocked record. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Input

and the DCB specifies:

- GET
- Exchange buffering
- Locate operating mode
- Unblocked record format (fixed-, variable-, or undefined-length).

The module consists of a GET routine and a RELSE routine.

The GET routine operates as follows:

- It receives control when a GET macro-instruction is encountered in a processing program.
- It passes control to the end-of-block routine to cause scheduling of the previous buffer for refilling.
- It passes control to the Input Synchronizing and Error Processing routine (module IGG019AQ) to obtain the next full buffer. (When the OPEN executor primes the buffers, it schedules all buffers except one.)
- It presents the address of the record, and returns control, to the processing program. For variable- or undefined-length records, the routine also presents the record length.

The RELSE routine returns control without performing any processing.

Note: If an input DCB using this module is paired with an output DCB using module IGG019EE (Output, PUT, Exchange), a PUTX macro-instruction addressed to the output DCB causes an exchange of the addresses of the current buffer segments of each DCB. These addresses are found in the CCWs pointed to by the DCBCCCW fields in the input and output DCBs.

GET Module IGG019EC

Module IGG019EC uses the substitute mode to present the processing program with the address of the next unblocked record. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Input

and the DCB specifies:

- GET
- Exchange buffering
- Substitute operating mode
- Unblocked record format (fixed-, or undefined-length).

The module consists of a GET routine and a RELSE routine.

The GET routine operates as follows:

- It receives control when a GET macro-instruction is encountered in a processing program.
- It passes control to the synchronizing routine to obtain the next full buffer.
- It exchanges the address of the work area and the address of the buffer.
- It passes control to the end-of-block routine to cause the work area offered by the processing program to be scheduled for filling.
- It presents the address of the new record, and returns control, to the processing program. (When the OPEN executor primes the buffers, it schedules all buffers.)
- When undefined-length records are specified, the routine also presents the record length.

The RELSE routine returns control without performing any processing.

GET Module IGG019ED

Module IGG019ED uses the substitute mode to present the processing program with the address of the next fixed-length blocked record. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Input

and the DCB specifies:

- GET
- Exchange buffering
- Substitute operating mode
- Fixed-length blocked record format.

The module consists of a GET routine and a RELSE routine.

The GET routine operates as follows:

- It receives control when a GET macro-instruction is encountered in a processing program.
- It tests for an end-of-block condition to determine if a new full buffer is needed. (When the OPEN executor primes the buffers, it schedules all buffers and sets an end-of-block condition.)
- If no end-of-block condition exists, it exchanges the address of the work area for the address stated in the current CCW. The current CCW is found by adding 8 to the value of the field DCBCCCW.

- If an initial end-of-block condition exists, it passes control to the Input Synchronizing and Error Processing routine (module IGG019AQ) to obtain the next full buffer. It then exchanges the address of the work area for the address stated in the first Read CCW of the channel program.

- It tests for a new end-of-block condition to determine if a buffer is empty and ready for refilling.

- If no new end-of-block condition exists, it presents the address of the next record, and returns control, to the processing program.

- If a new end-of-block condition exists, it passes control to the end-of-block routine to cause scheduling of the empty buffer for refilling. It then presents the address of the next record, and returns control, to the processing program.

The RELSE routine sets an end-of-block condition and passes control to the end-of-block routine to cause scheduling of the buffer for refilling. It then returns control to the processing program.

UPDATE MODE GET ROUTINE

The Update mode GET routine differs from other GET routines in that it shares its buffers (as well as the DCB and the IOBs) with the Update mode PUT routine. The QSAM Update mode of access uses simple buffering (in which the buffer is defined by the start and end address of the buffer).

If a PUTX macro-instruction addressed a record in a block, the Update mode GET routine determines, when the end of the block is reached, that that buffer is to be emptied (that is, that the block is to be updated) before being filled with a new block of data. If no PUTX macro-instruction addressed a record in a block, the Update mode GET routine determines, when the end of the block is reached, that the buffer is to be refilled only, that is, that the last block need not be updated and the buffer can be filled with a new block of data. These characteristics of the buffer, simple buffering, sharing the buffer with the PUT routine, and emptying the buffer before refilling, influence the manner in which the Update mode GET routine determines:

- The address of the next record.

- Whether the buffer can be scheduled.

- Whether a new buffer is needed.

- Whether to schedule the buffer for Empty-and-Refill or for Refill-only.

The first three of these determinations are made at every pass through the routine. The last determination is made after the routine establishes that the buffer can be scheduled.

If the records are unblocked, the address of the next record is the address of the next buffer.

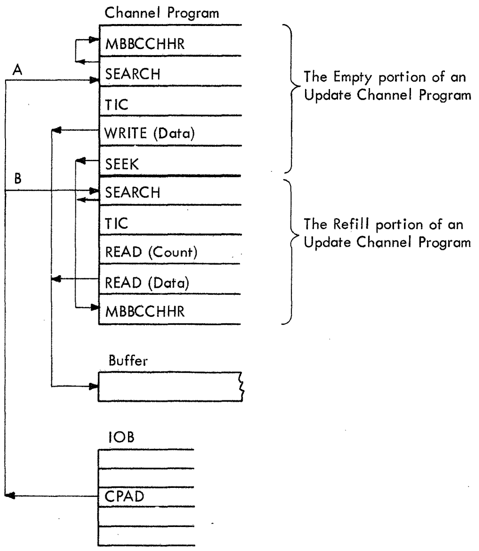
If the records are blocked, the address of the next record is found by adding the record length (found in the DCBLRECL field) to the value in the DCBRECAD field.

Whether the buffer can be scheduled and whether a new buffer is needed is determined by whether an end-of-block condition exists. In the Update mode, one determination that an end-of-block condition exists causes both the last buffer to be scheduled and a new buffer to be sought. An end-of-block condition exists for unblocked records at every pass through the routine; for blocked records it exists if the values in the DCBRECAD (the address of the current record) field and the DCBEOBAD (the address of the end of the block) field are equal. To cause scheduling of the buffer, the GET routine passes control to the end-of-block routine. To obtain a new buffer, the GET routine passes control to the Update Synchronizing and Error Processing routine (module IGG019AF).

To cause scheduling of the buffer for either Empty-and-Refill or Refill-only, the Update mode GET routine sets the IOB to point to the beginning of either of the two parts of QSAM Update channel program. These two parts are the Empty part, which empties (writes out of) the buffer, and the Refill part, which refills (reads into) that same buffer. (See Figure 4.) If execution of a QSAM Update channel program begins with the Empty part, it is always followed by execution of the Refill part. Each part of the QSAM Update channel program addresses a different location in auxiliary storage: The Empty part addresses the location from which the block to be updated was read; the Refill part addresses the location from which the last block was read. Addressing the last known block and skipping over its data field leads to the beginning of the next block, irrespective of its address. (This method of addressing a Search command to the block read previously to address a Read (Count, Key, and Data) command to the next block is known as the search-previous technique. It makes the count field of the present block being read the Seek address of the Refill portion of the next channel program.) When a buffer is to be emptied (back to the original location of the block in auxiliary storage), the Update mode GET routine obtains the block address from the Seek address of the Refill part of the next channel program. It copies the address so that it becomes the Seek address for the Empty part of the present channel program. (See Figure 5.) (For a description of the processing for a Refill-only QSAM Update channel program, refer to the description of the Update SIO appendage.)

Whether to schedule the buffer for Empty-and-Refill or for Refill-only depends on whether the block is to be updated. If the block is to be updated, the PUTX routine will have set the Update flag on in the IOB; else the flag is off. To schedule the buffer for Empty-and-Refill, the GET routine sets the IOB to point to the Empty portion of the channel program and obtains the Seek address of the block to be updated from the Refill portion of the next channel program. To schedule the buffer for Refill-only, the GET routine sets the IOB to point to the Refill portion of the channel program. The end-of-block condition which triggers this processing also causes control to pass to the end-of-block routine (module IGG019CC) for issuing the EXCP macro-instruction and to the Update Synchronizing and Error Processing routine (module IGG019AF) for obtaining the next buffer.

The PUTX routine sets the Update flag in the IOB and returns control to the processing program.

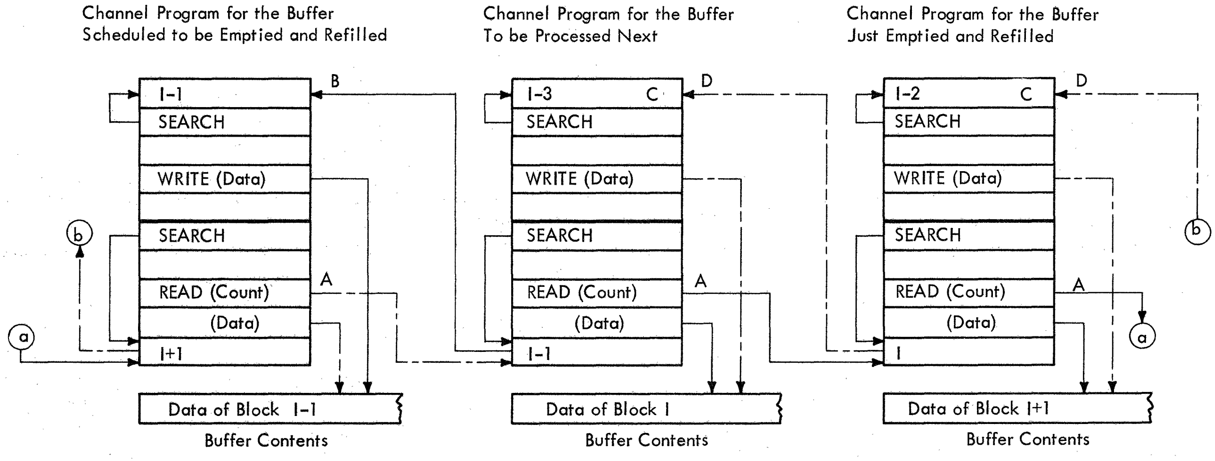


Legend:

A - Address of channel program (CPAD) used to empty and refill the buffer. (A PUTX macro-instruction was addressed to a record in this buffer.)

B - Address of channel program (CPAD) used only to refill the buffer. (No PUTX macro-instruction was addressed to any record in this buffer.)

Figure 4. The Two Parts of an Update Channel Program (Empty, Refill)



Legend:

A - The Refill portion reads the count field of the block being read into the search argument of the next Refill portion.

B - To empty the buffer, the search argument of the next Refill portion is used as the search argument of this Empty portion.

C - To empty the buffer, the search argument of the next Refill portion was copied before the last time this buffer was scheduled.

D - To empty the buffer, the search argument of the next Refill portion will be copied before the next time this buffer is scheduled.

———— Present entries
–·–·–·– Future entries

Figure 5. Relation of Seek Addresses in Three Successive QSAM Update Channel Programs

The RELSE routine sets an end-of-block condition and returns control to the processing program.

The OPEN executor primes (that is, schedules for filling) all the buffers except one if QSAM is used with a DCB opened for Update. The OPEN executor also sets an end-of-block condition; the first time that the Update mode GET routine gains control, it processes this condition in its normal manner.

There is one Update mode GET routine. If the access conditions shown in Table 4 are specified for a DCB, the OPEN executor selects this routine, loads it, and places its address into the DCBGET field.

Table 4. Module Selector - Update Mode GET Module

Access Conditions	Selections				
Update, GET	X	X	X	X	X
Fixed-length record format	X	X			
Variable-length record format			X	X	
Undefined-length record format					X
Blocked record format	X		X		
Unblocked record format		X		X	X
GET Module					
IGG019AE[1]	X	X	X	X	X

[1] This module also carries the Update mode PUTX routine

GET Module IGG019AE

Module IGG019AE presents the processing program with the next input record, flags the IOB if the block is to be updated (that is, emptied and refilled), and sets the IOB to address a QSAM Update channel program for either Empty-and-Refill or Refill-only. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Update

and the DCB specifies:

- GET.

The module consists of a GET routine, a RELSE routine, and a PUTX routine.

The GET routine operates as follows:

- It receives control when a GET macro-instruction is encountered in a processing program.
- It tests for an end-of-block condition to determine whether the buffer can be scheduled and whether a new buffer is needed. (When the OPEN executor primes the buffers, it schedules all buffers except one and sets an end-of-block condition.)
- If no end-of-block condition exists, it presents the address of the next record, and returns control, to the processing program. For variable-length and undefined-length records, it also determines the length of the record and places it in the DCBLRECL field in the DCB.
- If an end-of-block condition exists, it tests whether the buffer is to be emptied and refilled or is to be refilled only.
- If it is to be refilled only, it sets the IOB to point to the start of the Read portion of the Update channel program and passes control to the end-of-block routine to cause scheduling of the buffer.
- If it is to be emptied and refilled, it sets the IOB to point to the start of the Update channel program. The routine obtains the auxiliary storage address to be used by the Write portion of the channel program by copying the address used by the Read portion of the channel program associated with the next IOB. The routine then passes control to the end-of-block routine to cause scheduling of the buffer.
- On return of control from the end-of-block routine, the GET routine passes control to the Update Synchronizing and Error Processing routine (module IGG019AF) to obtain a new full buffer.
- On return of control from the synchronizing routine, the GET routine updates the DCBLRECL field and presents the address of the next record, and returns control, to the processing program.

The RELSE routine operates as follows:

- It receives control when a RELSE macro-instruction is encountered in the processing program.

- It sets an end-of-block condition.

- It returns control to the processing program.

The PUTX routine operates as follows:

- It receives control when a PUTX macro-instruction is encountered in the processing program.

- It sets the Update flag in the IOB to show that the buffer is to be emptied before being refilled.

- It returns control to the processing program.

PUT ROUTINES

There are seven different PUT routines. A particular PUT routine is used with a specific data set on the basis of the access condition options specified by the processing program for access to that data set.

A PUT routine gains CPU control when a PUT, PUTX, or TRUNC macro-instruction is encountered. The PUT routine returns control to the processing program, unless either an output buffer is ready to be scheduled for emptying or a new empty buffer is needed. If a buffer is ready for emptying, the PUT routine passes control to an end-of-block routine. If a new empty output buffer is needed, the PUT routine passes control to a synchronizing and error processing routine. A PUT routine accepts a record from the processing program to assemble a block of data for an output channel program. A PUTX routine accepts an output record from an input data set; a RELSE routine causes the present buffer to be considered ready for scheduling.

Every PUT routine determines in each pass through the routine:

- The address of the next buffer segment.

- Whether, an output buffer is to be scheduled for emptying.

- Whether a new empty output buffer is needed.

In each entry into a PUT routine, it accepts a record for output.

PUT routines differ mainly in the buffering techniques they support. PUT routines for simple buffering deal with buffers that are permanently associated with one DCB. PUT routines for exchange buffering deal with buffers that are exchanged between the output DCB, the input DCB, and the processing program. The PUTX routine for the Update mode of OPEN uses simple buffering; it differs from other PUT routines in that it shares the buffer used by the Update mode GET routine. The manner in which a PUT routine performs its processing depends on the buffering mode.

Simple buffering PUT routines determine the address of the next buffer segment by referring to the DCB. To determine whether a buffer is ready for scheduling and whether a new buffer is needed, these routines compare the beginning and ending address of the buffer (or the record and the remaining space in the buffer). To accept a record, a PUT routine using simple buffering either moves the record into the buffer or requires the processing program to do so.

Exchange buffering PUT routines determine the address of the next buffer segment by referring to the channel program. To determine whether a buffer is to be scheduled and whether a new buffer is needed, these routines compare the beginning and ending address of the channel program. To accept a record, an exchange buffering PUT routine exchanges its buffer segment for a work area or for a buffer segment of an input DCB, or may move the record into the buffer segment.

The Update mode PUTX routine flags the buffer from which the last record was presented for updating.

The PUT routine descriptions are accordingly grouped as:

- Simple Buffering PUT Routines.

- Exchange Buffering PUT Routines.

- Update Mode PUTX Routine.

SIMPLE BUFFERING PUT ROUTINES

Simple buffering PUT routines use buffers whose beginning and ending addresses are stated in the DCB. The beginning address is in the field DCBRECAD (address of the next record); the ending address is in the field DCBEOBAD (address of the end of the buffer). In each pass through a routine, it determines:

- The address of the next buffer segment.

- Whether an output buffer is to be scheduled for emptying.

- Whether a new empty buffer is needed.

These three determinations are made at every pass through a PUT routine.

If the records are unblocked, the address of the next available buffer segment is always that of the next buffer.

If the records are blocked, a PUT routine determines the address of the next available buffer segment by adding the length of the last record to the address of the last buffer segment. The address of the last buffer segment is in the DCBRECAD field of the data control block (DCB). If the records are fixed-length blocked records, the length of each record is in the DCBLRECL field. If the records are variable-length blocked records, the length of each record is in the length field of the record itself.

A PUT routine determines that a buffer is ready for emptying, and that a new empty buffer is needed, by establishing that an end-of-block (EOB) condition exists.

If an output buffer is to be scheduled for emptying, a PUT routine passes control to an end-of-block routine, to cause the present buffer to be scheduled for output.

If a new empty buffer is needed, a PUT routine obtains a new buffer by passing control to the Output Synchronizing and Error Processing routine (module IGG019AR). For a buffer that was emptied without error, the synchronizing routine updates the DCBIOBA field (thus pointing to the new buffer) and returns control to the PUT routine. The PUT routine updates the DCBRECAD field by inserting the starting address of the buffer from the channel program associated with the new IOB. To update the DCBEOBAD field, the routine adds the length of the block stated in the DCBBLKSIZE field to the buffer starting address. These two fields, DCBRECAD and DCBEOBAD, define the available buffer.

An EOB condition is established by different criteria for different record formats and operating modes.

For unblocked records, an EOB condition exists after each record is placed in the buffer. If using the move operating mode, a PUT routine establishes that an EOB condition exists for the present buffer after the routine has moved the record into the buffer. If using the locate operating mode, a PUT routine establishes that an EOB condition exists for the present buffer on the next entry to the routine, after the processing program has moved the record into the buffer.

For blocked records, the time that an EOB condition occurs depends on the record format.

For fixed-length blocked records, an EOB condition occurs when the DCBRECAD field equals the DCBEOBAD field. (The DCBRECAD field shows the address of the segment for the next record. The DCBEOBAD field shows a value equal to one more than the address of the end of the buffer.) If using the move operating mode, the PUT routine moves the last fixed-length record into the buffer, updates the DCBRECAD field, and establishes that an EOB condition exists for the present buffer. If using the locate operating mode, the processing program moves the last fixed-length record into the buffer. On the next entry to the PUT routine, the routine updates the DCBRECAD field, and establishes that an EOB condition exists for the present buffer.

For variable-length blocked records, an EOB condition occurs when the next record exceeds the buffer balance, that is, the record length is greater than the space remaining in the buffer. If using the move operating mode, the PUT routine establishes that an EOB condition exists when the record length stated in the first word of the record exceeds the buffer balance. If using the locate operating mode, the PUT routine establishes that an EOB condition exists when the value stated in the DCBLRECL field in the DCB exceeds the buffer balance.

A TRUNC routine sets an end-of-block condition to empty the buffer. This end-of-block condition is processed so that the next entry to the PUT routine permits it to operate as usual. Successive entries to a TRUNC routine without intervening entries to a PUT routine cause the TRUNC routine to return control without performing any processing.

To permit a PUT routine to operate normally when it is entered for the first time, the OPEN executor initializes the DCB fields DCBRECAD and DCBEOBAD. For an output data set using QSAM and simple buffering, the values entered in these fields depend on the operating mode. For locate mode routines, it sets them to show the beginning and end of the first buffer; for move mode routines it sets an end-of-block condition.

There are four simple buffering PUT routines. (Modules for the move operating mode include PUTX routines.) Table 5 lists the routines available and the conditions that cause a particular routine to be used. The OPEN executor selects one of the routines, loads it, and places its address into the DCBPUT field. The table shows, for example, that when the DCB specifies the locate mode and fixed-length records, routine IGG019AI is selected and loaded.

Table 5. Module Selector - Simple Buffering PUT Modules

Access Conditions	Selections					
Output, PUT/PUTX, Simple buffering	X	X	X	X	X	X
Locate operating mode	X	X	X			
Move operating mode				X	X	X
Fixed-length record format	X			X		
Undefined-length record format		X			X	
Variable-length record format			X			X
PUT Modules						
IGG019AI	X	X				
IGG019AJ			X			
IGG019AK				X	X	
IGG019AL						X

PUT Module IGG019AI

Module IGG019AI presents the processing program with the address of the next available buffer segment for a fixed-length or undefined-length record. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Output

and the DCB specifies:

- PUT
- Simple buffering
- Locate operating mode
- Fixed-length (unblocked, blocked, or blocked standard) or undefined-length record format.

The module consists of a PUT routine and a TRUNC routine.

The PUT routine operates as follows:

- It receives control when a PUT macro-instruction is encountered in a processing program.
- It determines the address of the next buffer segment using the value in the DCBLRECL field.
- It tests for an EOB condition to determine whether a buffer is full and ready for emptying and also whether a new empty buffer is needed.
- If no EOB condition exists, it presents the address of the next buffer segment to the processing program, and returns control to the processing program.
- If an EOB condition exists, it issues a BALR instruction to pass the present buffer to the end-of-block routine. The PUT routine issues another BALR instruction to obtain a new buffer through the Output Synchronizing and Errcr Processing routine (module IGG019AR), and determines the address of the first segment of the new buffer. The PUT routine then presents this address to the processing program and returns control to the processing program.

The TRUNC routine causes an EOB condition ty setting the DCBRECAD and DCBEOBAD fields so that they are equal; it then returns control to the processing program.

PUT Module IGG019AJ

Module IGG019AJ presents the processing program with the address of the next available buffer segment for a variable-length record. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Output

and the DCB specifies:

- PUT
- Simple buffering
- Locate operating mode
- Variatle-length (unblocked, blocked) record format.

The module consists of a PUT routine and a TRUNC routine.

The PUT routine operates as follows:

- It receives control when a PUT macro-instruction is encountered in a processing program.

- It determines the address of the next buffer segment using the length field of the record moved by the processing program into the buffer segment located last.

- It tests for an EOB condition to determine whether a buffer is ready for emptying and also whether a new empty buffer is needed, using the value placed into the DCBLRECL field by the processing program.

- If no EOB condition exists, it tests for blocked records.

- If blocked records are specified, it presents the address of the next buffer segment to the processing program, and returns control to the processing program.

- If an EOB condition exists or if unblocked records are specified, it issues a BALR instruction to pass the present buffer to the end-of-block routine. The PUT routine issues another BALR instruction to obtain a new buffer through the Output Synchronizing and Error Processing routine (module IGG019AR), and determines the address of the first segment of the new buffer. The PUT routine then presents this address to the processing program and returns control to the processing program.

The TRUNC routine causes an EOB condition by setting the DCBRECAD and DCBEOBAD fields so that they are equal; it then returns control to the processing program.

PUT Module IGG019AK

Module IGG019AK moves the present fixed-length or undefined-length record into the next available buffer segment. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Output

and the DCB specifies:

- PUT
- Simple buffering
- Move operating mode
- Fixed-length (unblocked, blocked, blocked standard) or undefined-length record format.

The module consists of a PUT routine, a PUTX routine, and a TRUNC routine.

The PUT routine operates as follows:

- It receives control when a PUT macro-instruction is encountered in a processing program.

- If an EOB condition exists, it issues a BALR macro-instruction to obtain a new buffer through the Output Synchronizing and Error Processing routine (module IGG019AR), and then moves the record from the work area into the first buffer segment.

- If no EOB condition exists, it moves the record from the work area into the next buffer segment.

- It tests for blocked records.

- If blocked records are specified, it determines the address of the next segment and tests for a new EOB condition.

- If unblocked records are specified or if a new EOB condition exists, it issues a BALR instruction to pass the present buffer to the end-of-block routine, and then returns control to the processing program.

- If no new EOB condition exits, it returns control to the processing program.

The PUTX routine operates as follows:

- It receives control when a PUTX macro-instruction is encountered in a processing program.

- It obtains the DCBRECAD value of the input DCB, which points to the present record in the input buffer.

- It enters the PUT routine at the start. The PUT routine then uses the input DCBRECAD value in place of the work area address.

The TRUNC routine operates as follows:

- It receives control when a TRUNC macro-instruction is encountered in a processing program.

- It simulates an EOB condition.

- It issues a BALR instruction to pass the present buffer to the end-of-block routine.

- On return of control from the end-of-block routine it returns control to the processing program.

PUT Module IGG019AL

Module IGG019AL moves the present variable-length record into the next available buffer segment. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Output

and the DCB specifies:

- PUT
- Simple buffering
- Move operating mode
- Variable-length (unblocked or blocked) record format.

The module consists of a PUT routine, a PUTX routine, and a TRUNC routine.

The PUT routine operates as follows:

- It receives control when a PUT macro-instruction is encountered in a processing program.

- It determines the address of the next buffer segment and compares the length of the next record with the remaining buffer capacity.

- If the record fits into the buffer, it moves the record, updates the length field of the block, and tests for blocked records.

- If blocked records are specified, it returns control to the processing program.

- If the record does not fit into the buffer or if unblocked records are specified, it issues a BALR instruction to pass the present buffer to the end-of-block routine. It issues another BALR instruction to obtain a new buffer through the Output Synchronizing and Error Processing routine (module IGG019AR). The PUT routine then moves the record from the work area to the buffer, updates the block-length field, and returns control to the processing program.

The PUTX routine operates as follows:

- It receives control when a PUTX macro-instruction is encountered in a processing program.

- It obtains the DCBRECAD value of the input DCB, which points to the present record in the input buffer.

- It enters the PUT routine at the start. The PUT routine then uses the input DCBRECAD value in place of the work area address.

The TRUNC routine operates as follows:

- It receives control when a TRUNC macro-instruction is encountered in a processing program.

- It issues a BALR instruction to pass control of the present buffer to the end-of-block routine.

- It issues another BALR instruction to obtain a new buffer through the Output Synchronizing and Error Processing routine (module IGG019AR).

- It determines the address of the first segment of the new buffer, and then returns control to the processing program.

EXCHANGE BUFFERING PUT ROUTINES

Exchange buffering PUT routines use buffers whose addresses and lengths are in the channel program. For unblocked records, a buffer address and length are in one channel command word (CCW). For blocked records, addresses of buffer segments are in successive CCWs (though the segments themselves are not necessarily located next to one another). In each pass through an exchange buffering GET routine, it determines:

- The address of the next buffer segment.

- Whether an output buffer is to be scheduled for emptying.

- Whether a new empty buffer is needed.

These three determinations are made at every pass through a PUT routine.

If the records are unblocked, a PUT routine finds the address of the next buffer in the Write CCW for the next buffer.

If the records are blocked, a PUT routine finds the address of the next buffer segment in the next Write CCW. The next CCW is found by adding 8 to the address of the previous CCW, the value in the DCB field DCBCCCW.

If an output buffer is to be scheduled for emptying, a PUT routine passes control to an end-of-block routine to cause scheduling of the buffer. An end-of-block routine passes control to the I/O supervisor to have it schedule the buffer. After scheduling, the I/O supervisor returns control to the end-of-block routine, and it returns control to the PUT routine.

If a new empty buffer is needed, a PUT routine passes control to the Output synchronizing and error processing routine. If the channel program for the next buffer has been executed without error, the synchronizing routine enters the address of the input/output block (IOB) that points to that channel program into the DCBIOBA field in the DCB.

An output buffer is to be scheduled for emptying and a new buffer is needed if an end-of-block condition exists. When using exchange buffering with an output data set, the buffer can be scheduled for emptying when the address of the last record has been placed in the last CCW or a record has been moved into the last segment. Accordingly, an end-of-block test is made before leaving the routine. This test determines whether the buffer is to be scheduled. another test is made on entry to determine whether a new buffer is needed. An end-of-block condition exists for unblocked records each time the routine is entered; for blocked records it exists if the address of the current CCW (in field DCBCCCW) and the address of the last CCW (in field DCBLCCW) are the same.

A TRUNC routine sets an end-of-block condition to empty the buffer. This end-of-block condition is processed so that the next entry to the PUT routine permits it to operate as usual. Successive entries to a TRUNC routine without intervening entries to a PUT routine cause the TRUNC routine to return control without performing any processing.

The processing performed by the OPEN executor for an output data set using QSAM and exchange buffering includes setting an end-of-block condition. On the first entry to an exchange buffering PUT routine it processes this condition as usual.

There are two exchange buffering PUT routines. Table 6 lists each of these routines and the conditions that cause either routine to be used. The OPEN executor selects one of the routines, loads it, and places its address into the DCBPUT field. The table shows, for example, that if Output, PUT, exchange, move, and unblocked record format are specified, module IGG019EE is selected for use as the PUT routine.

Table 6. Module Selector - Exchange Buffering PUT Modules

Access Conditions	Selections						
Output, PUT/PUTX, Exchange	X	X	X	X	X	X	X
Move mode	X	X	X			X	
Substitute mode				X	X		X
Unblocked record format	X	X	X	X	X		
Blocked record format						X	X
Fixed-length record format	X			X		X	X
Variable-length record format		X					
Undefined-length record format			X		X		
PUT Modules							
IGG019EE	X	X	X	X	X		
IGG019EF						X	X

PUT Module IGG019EE

Module IGG019EE puts an unblocked record into the next buffer. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Output

and the DCB specifies:

- PUT, PUTX
- Exchange buffering
- Unblocked record format
- Move operating mode and fixed-, variable-, or undefined-length record format; or substitute operating mode and fixed-, or undefined-length record format.

The module consists of a PUT routine, a PUTX routine, and a TRUNC routine.

The PUT routine operates as follows for the Move mode:

- It receives control if a PUT macro-instruction is encountered in the processing program.
- It passes control to the Output Synchronizing and Error Processing routine (module IGG019AR) to obtain the next buffer.
- It determines the address of the Write (data) CCW, enters the length in the CCW and finds the buffer address.
- It moves the record from the work area into the buffer.
- It passes control to the end-of-block routine to cause scheduling of the buffer.
- It returns control to the processing program.

The PUT routine operates as follows for the Substitute mode:

- It receives control when a PUT macro-instruction is encountered in a processing program.
- It passes control to the Output Synchronizing and Error Processing routine (module IGG019AR) to obtain the next buffer.
- It determines the address of the Write (data) CCW, enters the length in the CCW and finds the buffer address.
- It exchanges the address of the work area and the address of the buffer area.
- It passes control to the end-of-block routine to cause scheduling of the buffer for output.
- It returns control, and the address of the buffer, to the processing program.

The PUTX routine operates as follows if the input DCB specifies simple buffering:

- It receives control when a PUTX macro-instruction is encountered in a processing program.
- It passes control to the Output Synchronizing and Error Processing routine (module IGG019AR) to obtain the next buffer.
- It finds the address of the input buffer in the DCBRECAD field of the input DCB and the input buffer length in the DCBLRECL field.
- It moves the record from the input buffer to the output buffer and enters the length in the Write (data) CCW.
- It passes control to the end-of-block routine to cause scheduling of the buffer for output.
- It returns control to the processing program.

The PUTX routine operates as follows if the input DCB specifies exchange buffering:

- It receives control when a PUTX macro-instruction is encountered in a processing program.
- It passes control to the Output Synchronizing and Error Processing routine (module IGG019AR) to obtain the next buffer.
- It finds the address of the Read CCW and the length of the buffer in the DCBCCCW and DCBLRECL fields of the input DCB; it finds the address of the Write CCW in the DCBCCCW field of the output DCB.
- It exchanges the buffer addresses and enters the length into the Write CCW.
- It passes control to the end-of-block routine to cause scheduling of the buffer for output.
- It returns control to the processing program.

The TRUNC routine receives control when a TRUNC macro-instruction is encountered in a processing program; it returns control to the processing program without performing any processing.

PUT Module IGG019EF

Module IGG019EF puts a blocked record into the next buffer segment. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Output

and the DCB specifies:

- PUT, PUTX
- Exchange buffering
- Move or substitute operating mode
- Fixed-length blocked record format.

The module consists of a PUT routine, a PUTX routine, and a TRUNC routine.

The PUT routine operates as follows:

- It receives control when a PUT macro-instruction is encountered in the processing program.
- If there is an end-of-block condition on entry to the routine, it passes control to the Output Synchronizing and Error Processing routine (module IGG019AR) to obtain the next buffer.
- If the move mode is used, and either there is no end-of-block condition or control has returned from the synchronizing routine, the PUT routine moves the record from the work area into the next buffer segment.
- If the substitute mode is used, and either there is no end-of-block condition or control has returned from the synchronizing routine, the PUT routine exchanges the current buffer segment address of the output DCB for either the current buffer segment address of the input DCB or the address of a work area.
- It tests for another end-of-block condition to determine if the buffer is to be scheduled for output.
- If there is no end-of-block condition, it returns control to the processing prcgram.
- If there is an end-of-block condition, it passes control to the end-of-block routine to cause scheduling of the buffer. On return of control to the PUT routine, it returns control to the processing program.

The PUTX routine operates as follows:

- It receives control when a PUTX macro-instruction is encountered in the processing program.
- If there is an end-of-block condition on entry to the routine, it passes control to the Output Synchronizing and Error Processing routine (module IGG019AR) to obtain the next buffer.
- If the input DCB uses simple buffering, and either there is no end-of-block condition or control has returned from the synchronizing routine, the PUTX routine moves the record from the input buffer segment into the next output buffer segment.
- If the input DCB uses exchange buffering, and either there is no end-of-block condition or control has returned from the synchronizing routine, the PUTX routine exchanges the buffer segment addresses of the current output and input CCWs.
- It tests for another end-of-block condition to determine if the buffer is to be scheduled for output.
- If there is no end-of-block condition, it returns control to the processing program.
- If there is an end-of-block condition, it passes control to the end-of-block routine to cause scheduling of the buffer for output. On return of control to the PUTX routine, it then returns control to the processing program.

The TRUNC routine operates as follows:

- It receives control when a TRUNC macro-instruction is encountered in a processing program.
- It returns control to the processing program without any further processing if the buffer was scheduled for output on the preceding entry into the PUT or PUTX routine.
- It turns off the chain-data bit in the CCW used in the preceding pass through the PUT or PUTX routine. (The chain-data bit is set on in every CCW in the normal course of operation of the PUT or PUTX routine to offset any possible prior truncation.)

- It passes control to the end-of-block routine to cause scheduling of the buffer for output. On return of control, the TRUNC routine then returns control to the processing program.

UPDATE MODE PUTX ROUTINE

The Update mode PUTX routine differs from other PUT routines in that it shares its buffers (as well as the DCB and the IOBs) with the Update mode GET routine. It is the Update mode GET routine that determines the address of the segment, when the end of the buffer is reached, and when a new buffer is needed. Thus all that is left for the PUTX routine to do is to flag the block for output.

There is one Update mode PUT routine; it is part of module IGG019AE which also carries the Update mode GET routine. The module (including the PUTX routine) is described in the Update mode GET routine section of this manual. \

END-OF-BLOCK ROUTINES

There are nine different end-of-block routines. They are selected for use with a particular data set on the basis of the access conditions specified by the processing program for that data set. Unless Inout or Outin is specified in the OPEN parameter list, one end-of-block routine is selected. If Inout or Outin are specified, two end-of-block routines may be required.

An end-of-block routine receives control from a GET or a PUT routine (when using QSAM), or from a READ or WRITE routine (when using BSAM). In general, end-of-block routines pass control to the I/O supervisor. An end-of-block routine receives control from a GET or a PUT routine when a buffer is ready for scheduling. An end-of-block routine receives control from a READ or WRITE routine at each pass through those routines. Control passes from an end-of-block routine to the I/O supervisor, except when a channel program is chained to another one not yet executed. End-of-block routines provide device oriented entries for the channel program, such as control characters and auxiliary storage addresses.

End-of-block routine descriptions are grouped as follows:

- Ordinary end-of-block routines. These routines perform device oriented processing when normal channel program scheduling is used (except when it is used with an output data set with track overflow).

- Chained channel-program scheduling end-of-block routines. These routines perform device oriented processing and attempt to chain channel programs when chained channel-program scheduling is used.

- Track overflow end-of-block routine. This routine performs device oriented processing and computes segment lengths and constructs count fields when track overflcw (which uses normal channel-program scheduling) is used with an output data set.

ORDINARY END-OF-BLOCK ROUTINES

Ordinary end-of-block routines process channel programs for all devices. This processing is independent of the progress of a previous channel program and causes access to proceed one channel program at a time. In the case of output data sets on direct-access devices, the routines limit the size of the block to the track capacity. For direct-access devices, an ordinary end-of-block routine computes auxiliary storage addresses fcr output data sets and input data sets with fixed-length standard record format to avoid end-of-track interruptions. For unit record devices these routines process control characters and PRTOV macro-instructions. For an input data set with track overflow progression from track to track is controlled by the track overflow bit in the overflowing segment, not by computation of the end-of-block routine nor by an entry in the channel program.

There are four ordinary end-of-block routines. Table 7 lists the routines available and the conditions that cause a particular routine to be used. For QSAM, the OPEN executor selects one of the routines, loads it and places its address into the DCBEOB field. For BSAM and BPAM the OPEN executor selects one of the routines, loads it, and places its address into both the DCBEOBR and DCBEOBW fields. If Inout or Outin is specified, a second end-of-block routine may be selected and loaded. Its address replaces one of the duplicate addresses in the DCB. The table, for example, shows that when normal channel-program program scheduling is used, and the device type is magnetic tape, routine IGG019CC is selected and loaded for use as the end-of-block routine for that DCB.

Table 7. Module Selector - Ordinary End-of-Block Modules

Access Conditions	Selections									
Normal channel-program scheduling	X	X	X	X	X	X	X	X	X	X
Input, or				X	X	X				
Update			X							
Output, or							X			
Inout, Outin				X		X	X			
Card reader or paper tape reader	X									
Printer or card punch								X	X	X
Magnetic tape		X								
Direct-access storage				X		X	X			
Track overflow					X					
Record format is not fixed-length standard				X						
Record format is fixed-length standard						X				
No control character								X		
Machine control character									X	
ASA control character										X
PRTOV-No user exit								X	X	X
End-of-Block Modules										
IGG019CC	X	X	X	X	X					
IGG019CD						X	X			
IGG019CE								X	X	
IGG019CF										X

End-Of-Block Module IGG019CC

Module IGG019CC does nothing more than cause a channel program to be scheduled.

The OPEN executor selects and loads this module if one of the following conditions exists:

- The DCB specifies normal channel-program scheduling and magnetic tape, card reader, or paper tape as the device type.
- The data set is opened for Input, and the DCB specifies normal channel-program scheduling, direct-access storage device, and a record format other than fixed-length length standard.
- The data set is opened for Inout or Outin, and the DCB specifies normal channel-program scheduling, direct-access device storage and a record format other than fixed-length standard. The address of this module is placed in the DCBEOBR field.
- The data set is opened for Update.

The module operates as follows:

- It receives control when a GET or a PUT routine finds that a buffer is ready to be scheduled, or at the conclusion of the processing performed by a READ or WRITE routine.
- If the device type is magnetic tape, paper tape, or card reader, the module issues an EXCP macro-instruction and returns control to the GET, PUT, READ, or WRITE routine.
- If the device type is direct-access and more than one IOB is associated with the DCB, the module issues an EXCP macro-instruction and returns control to the GET or READ routine.
- If the device type is direct-access and only one IOB is associated with the DCB, the module copies the DCBFDAD field in the DCB into the IOBSEEK field in the IOB, issues an EXCP macro-instruction and returns control to the GET or READ routine.

End-of-Block Module IGG019CD

Module IGG019CD schedules a channel program after determining that the next block fits on a track within the allocated extents.

The OPEN executor selects and loads this module if one of the following conditions exists:

- The data set is opened for Output, and the DCB specifies normal channel-program scheduling, no track-overflow, and direct-access storage as the device type.

- The data set is opened for Input, and the DCB specifies normal channel-program scheduling, direct-access storage as the device type.

- The data set is opened for Inout or Outin, and the DCB specifies direct-access device storage. If the record format (also specified in the DCB) is other than fixed-length standard the address of this module is placed in the DCBEOBW field. If the record format is fixed-length standard, the address of this module is placed in both DCBEOBR and DCBEOBW fields.

The module operates as follows:

- It receives control when a GET or a PUT routine finds that a buffer is ready to be scheduled, or at the conclusion of the processing performed by a READ or WRITE routine.

- It calculates the block length using the value overhead-last record. (This value is found in the resident I/O device table. The address of the table is in the field DCBDVTBL.) It compares the calculated block length with the value in the DCBTRBAL field of the DCB.

- If the block length is equal to or less than the DCBTRBAL field value, the module determines that the block fits on the track.

- If the block length exceeds the DCBTRBAL field value, the module finds the next track as follows:

  It converts the full device address (MBBCCHHR) of the present track into a relative address (TTR) by passing control to the IECPRLTV routine.

  It adds 1 to the value of TT.

  It converts the relative address of the next track into the full device address by passing control to the IECPCNVT routine.

- If there is another track in the allocated extents, its full address has been entered in the field DCBFDAD and the block fits on the track.

- If there is no other track in the allocated extents (as shown by the error return code from routine IECPCNVT), an EOV condition exists. The module sets the DCBCIND1 field in the DCB and the CSW field in the IOB to show this, and returns control to the GET, PUT, READ, or WRITE routine without issuing an EXCP macro-instruction. The EOV condition is eventually recognized and processed, in QSAM by the synchronizing routine, in BSAM by the CHECK routine.

- When the module determines that the block fits on the track, the module calculates the actual block length, using the value overhead-not last record. (This value is found in the resident I/O device table.) It adjusts the value in the DCBTRBAL field by this amount, and updates the DCBFDAD field and the ID field of the count area of the block (located immediately after the channel program). It then issues an EXCP macro-instruction and returns control to the GET, PUT, READ, or WRITE module.

End-of-Block Module IGG019CE

Module IGG019CE, if necessary, modifies channel programs for unit record output devices when ASA control characters are not used. The module then causes scheduling of the channel program, whether it was modified or not. The OPEN executor selects ana loads this module if the DCB specifies:

- Normal channel-program scheduling
- Punch, or printer
- Machine control character, or no control character.

The module operates as follows:

- It receives control when a PUT routine finds that a buffer is ready to be scheduled, or at the conclusion of the processing performed by a WRITE routine.

- It adjusts, in the channel program, the length and starting address either for the length field of variable-length records or for a control character. If there are variable-length records and a control character, the module adjusts for both.

- If a control character is present, it inserts it as the command byte of the Write channel command word (CCW).

- It tests the DCB field at location (DCBDEVT+1) for a PRTOV mask. If a PRTOV mask is present, the module temporarily inserts it into the length field of the NOP CCW and sets the first bit in the IOB. The PRTOV appendage (IGG019CL) tests for the presence of the IOB bit and the CCW mask.

- It issues an EXCP macro-instruction and returns control to the PUT or WRITE routine.

<u>End-of-Block Module IGG019CF</u>

Module IGG019CF modifies channel programs for unit record output devices when an ASA control character is present. The module then causes scheduling of the channel program, whether it was modified or not. The OPEN executor selects and loads this module if the DCB specifies:

- Normal channel-program scheduling
- Punch, or printer
- ASA control character.

The module operates as follows:

- It receives control when a PUT routine finds that a buffer is ready to be scheduled, or at the conclusion of the processing performed by a WRITE routine.

- It adjusts, in the channel program, the length and starting address for the control character, and for the length field of variable-length records.

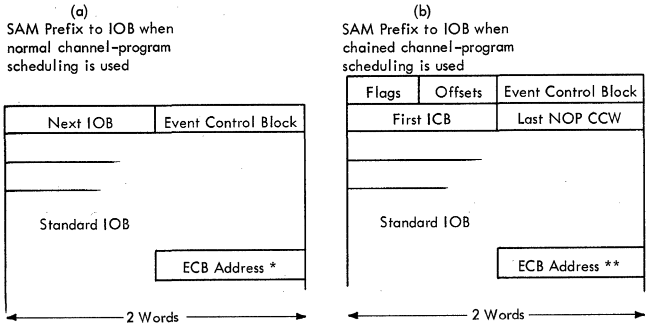
- It translates the control character and inserts it as the command byte of the control channel command word (CCW) which precedes the Write CCW.

- It tests the DCB field at location (DCBDEVT+1) for a PRTOV mask. If a PRTOV mask is present, the module inserts it into the length field of the Control CCW and sets the first bit in the IOB. The PRTOV appendage (IGG019CL) tests for the presence of the IOB bit and the CCW mask.

- It issues an EXCP macro-instruction and returns control to the PUT or WRITE routine.

CHAINED CHANNEL-PROGRAM SCHEDULING END-OF-BLOCK ROUTINES

Chained channel-program scheduling consists of joining the channel programs before execution and parting and posting the channel programs after execution. Joining is performed by the end-of-block routines and mainly uses the input/output block (IOB); parting and posting is performed by appendages and uses the interruption control block (ICB). (For a description of the parting process, refer to the program controlled interruption -PCI- appendages.) The IOB constructed by the OPEN executor when chained channel-program scheduling is used differs from the IOB used in normal channel-program scheduling. These differences are illustrated in Figure 6 and tabulated in Table 8.


(a)
SAM Prefix to IOB when normal channel-program scheduling is used

(b)
SAM Prefix to IOB when chained channel-program scheduling is used

* When QSAM is used, the address is that of the ECB in the SAM prefix; when BSAM is used the address is that of the ECB in the data event control block (DECB).

** Always shows the address of the ECB in the SAM prefix, irrespective of whether QSAM or BSAM is used.

Figure 6. Comparison of the IOB SAM Prefixes for Normal and for Chained Scheduling

These routines join channel programs so that the channel executes successive channel programs without interruption as if they were one continuous channel program. To join the present channel program to one already scheduled, the end-of-block routine finds the last CCW of the preceding channel program, by referring to the IOB, and changes that CCW from a NOP command to a TIC command. If this joining is performed before the channel attempts to execute (more precisely, before it fetches) that CCW, the joining process is successful. If the execution of the preceding channel program is completed while the routine is operating the joining is unsuccessful. The routine tests the success or failure of the joining by testing whether the IOB has been posted as completed. If successful, control returns to the calling program; if unsuccessful, the routine resets the IOB for the EXCP macro-instruction and passes control to the I/O supervisor.

The chained scheduling end-of-block routines, like the ordinary end-of-block routines, provide device oriented entries for channel programs. For direct-access devices they compute auxiliary storage addresses; for unit record devices they process control characters. (No processing is performed for the PRTOV macro-instruction since it and chained scheduling are mutually exclusive.)

There are four chained scheduling end-of-block routines, each performing joining and channel program entry processing for a different set of access condition options. Table 9 lists the routines available and the conditions that cause a particular routine to be used.

For QSAM, the OPEN executor selects one of the routines, loads it and places its address into the DCBEOB field. For BSAM and BPAM the OPEN executor selects one of the routines, loads it, and places its address into both the DCBEOBR and DCBEOBW fields. If Inout or Outin is specified, a second end-of-block routine may be selected and loaded. Its address replaces one of the duplicate addresses in the DCB.

The table, for example, shows that when chained scheduling is used, the Open mode is Input, and the device type is magnetic tape, routine IGG019CW is selected and loaded for use as the end-of-block routine for the DCB.

Table 8. Comparison of the IOB SAM Prefixes for Normal and for Chained Scheduling

Prefix Parameter	Normal Scheduling	Chained Scheduling
Number of IOBs	As many as there are buffers or channel programs	Only 1 (There are as many ICBs as there are buffers or channel programs)
Size of SAM Prefix	2 words	4 words
Contents of Link Address field	Address of the next IOB	Flags Offsets
Use of ECB field	Used in QSAM to post channel program execution (In BSAM, the ECB in the DECB is used)	Used in QSAM and BSAM to post a channel program execution that is terminated by channel end interruption (that is, channel program chaining has been broken)
Contents of IOBFICB field	Field does not exist	Address of the first ICB
Contents of IOBLNOP field	Field does not exist	Address of NOP CCW of channel program scheduled last

Table 9. Module Selector-Chained Channel-Program Scheduling End-of-Block Modules

Access Conditions	Selections							
Chained channel-program scheduling	X	X	X	X	X	X	X	X
Input	X	X		X				
Output			X		X	X	X	X
Card reader	X							
Printer or card punch						X	X	X
Magnetic tape		X	X					
Direct-access storage				X	X			
No control character						X		
Machine control character							X	
ASA control character								X
End-of-Block Modules								
IGG019CV					X			
IGG019CW	X	X	X	X				
IGG019CX						X	X	
IGG019CY								X

End-of-Block Module IGG019CV

Module IGG019CV computes from the track balance (and from further allocated extents on this volume, if necessary) a valid storage address for a channel program for an output data set on a direct-access device, and attempts to join the channel program to the preceding one. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Output

and the DCB specifies:

- Chained channel-program scheduling
- Direct-access storage.

The module operates as follows:

- It receives control from a PUT routine when that routine finds that a buffer is ready to be scheduled, or from a WRITE routine at the conclusion of its processing.
- It calculates the block length using the overhead value for a last block on a track. (This value is found in the resident I/O device table. The address of the table is in the field DCBDVTBL.) It compares the calculated block length with the value in the DCBTRBAL field of the DCB.
- If the block length is equal to or less than the DCBTRBAL field value, the module determines that the block fits on the track.
- If the block length exceeds the DCBTRBAL field value, the module calculates the next sequential track address and compares it with the end address of the current extent shown in the data extent block (DEB).
- If no end-of-extent condition exists, it determines that the block fits on the track.
- If an end-of-extent condition exists, it seeks a new extent in the DEB.
- If a new extent exists, it updates the DCBFDAD and DCBTRBAL fields and determines that the block fits on the track.
- If there is no further extent, an EOV condition exists. The module sets the DCBCIND1 field in the DCB and the CSW field in the IOB to show this, and returns control to the GET, PUT, READ, or WRITE routine without issuing an EXCP macro-instruction. The EOV condition is eventually recognized and processed, in QSAM by the synchronizing routine, in BSAM by the CHECK routine.
- If the module determines that the block fits on the track, the module calculates the actual block length using the overhead value for a block that is not the last on a track. (This value is found in the resident I/O device table.) It adjusts the value in the DCBTRBAL field by this amount, and updates the DCBFDAD field and the ID field of the count area of the block (located immediately after the channel program).

- If the block fits on the track, the module next attempts to join the channel program for the current buffer to the preceding channel program (that is, chain schedule) by:

  Setting the ICB to not-complete.

  Inserting the address of either the Write or the Search CCW of this channel program into the NOP CCW of the preceding channel program. The address of the Write CCW is inserted if the present and the preceding channel program address the same track. The address of the Search CCW is inserted if the present and the preceding channel programs address different tracks. In this case, the Search CCW addresses record zero of the next track.

  Changing the NOP CCW in the preceding channel program to a TIC CCW.

  Updating the SAM IOB prefix block to point to the end of the current channel program.

- It determines whether the joining was successful by testing the ECB (pointed to by the IOB) for a completion posting by the I/O supervisor.

- If the I/O supervisor did not post the event as completed, the joining was successful and the routine returns control to the calling routine.

- If the I/O supervisor did post the event as completed, the routine tests the ICB for the present channel program to find whether the joining was successful or not.

- If the present ICB remains unposted, the present channel program was not joined to the preceding one. The routine prepares to cause restart of the channel by copying the Seek address and the channel program start address from the current ICB into the IOB, and uses the EXCP macro-instruction to cause scheduling of the channel program. It then returns control to the calling routine.

- If the present ICB is posted complete, the present channel program was joined successfully. (The routine was interrupted long enough, between the joining and the testing, for the channel program to be executed and for the channel end appendage to post the ICB.) The routine returns control to the calling routine.

End-of-block Module IGG019CW

Module IGG019CW attempts to join the present channel program to the last one in the chain of scheduled channel programs. The OPEN executor selects and loads this module if either of the following conditions exists:

- The OPEN parameter list specifies Input and the DCB specifies chained channel-program scheduling and any device.

- The OPEN parameter list specifies Output and the DCB specifies chained channel program scheduling and magnetic tape.

The module operates as follows:

- It receives control from a GET or a PUT routine when that routine finds that a buffer is ready to be scheduled, or from a READ or WRITE routine at the conclusion of its processing.

- If the device type is magnetic tape, the routine determines the increment value and stores it in the ICB.

- The module attempts to join the channel program for the current buffer to the preceding channel program (that is, chain schedule) by:

  Setting the ICB to not-complete.

  Inserting the address of the current channel program into the NOP CCW of the preceding channel program.

  Changing the NOP CCW in the preceding channel program to a TIC CCW.

  Updating the SAM IOB prefix block to point to the end of the current channel program.

- It determines whether the joining was successful by testing the ECB (pointed to by the IOB) for a completion posting by the I/O supervisor.

- If the I/O supervisor did not post the event as completed, the joining was successful and the routine returns control to the calling routine.

- If the I/O supervisor did post the event as completed, the routine tests the ICB for the present channel program to find whether the joining was successful or not.

- If the present ICB remains unposted, the present channel program was not joined to the preceding one. The routine prepares to cause restart of the channel by copying the channel program start address (and the Seek address, if direct-access storage) from the current ICB into the IOB, and uses the EXCP macro-instruction to cause scheduling of the channel program. It then returns control to the calling routine.

- If the present ICB is posted complete, the present channel program was joined successfully. (The routine was interrupted long enough, between the joining and the testing, for the channel program to be executed and for the channel end appendage to post the ICB.) The routine returns control to the calling routine.

End-of-block Module IGG019CX

Module IGG019CX, if necessary, modifies channel programs for unit record output devices when ASA control characters are not used. The module then attempts to join the current channel program to the preceding one. The OPEN executor selects and loads this module if the DCB specifies:

- Chained channel-program scheduling
- Printer or card punch
- No control character, machine control character.

The module operates as follows:

- It receives control from a PUT routine when that routine finds that a buffer is ready for scheduling, or from a WRITE routine at the conclusion of its processing.

- It adjusts the length entry and the start address entry in the channel program for either a control character or a variable-length block length field or for both, if both are present.

- It inserts the control character, if present, as the command byte of the Write channel command word (CCW).

- It attempts to join the channel program for the current buffer to the preceding channel program (that is, chain schedule) by:

  Setting the ICB to not-complete.

  Inserting the address of the current channel program into the NOP CCW of the preceding channel program.

  Changing the NOP CCW in the preceding channel program to a TIC CCW.

  Updating the SAM IOB prefix block to point to the end of the current channel program.

- It determines whether the joining was successful by testing the ECB pointed to by the IOB for a completion posting by the I/O supervisor.

- If the I/O supervisor did not post the event as completed, the joining was successful and the routine returns control to the calling routine.

- If the I/O supervisor did post the event as completed, the routine tests the ICB for the present channel program to find whether the joining was successful or not.

- If the present ICB remains unposted, the present channel program was not joined to the preceding one. The routine prepares to cause restart of the channel by copying the channel program start address from the current ICB into the IOB, and uses the EXCP macro-instruction to cause scheduling of the channel program. It then returns control to the calling routine.

- If the present ICB is posted complete, the present channel program was joined successfully. (The routine was interrupted long enough, between the joining and the testing, for the channel program to be executed and for the channel end appendage to post the ICB.) The routine returns control to the calling routine.

End-of-Block Module IGG019CY

Module IGG019CY modifies channel programs for unit record output devices when ASA control characters are used. The module then attempts to join the current channel program to the preceding one. The OPEN executor selects and loads this module if the DCB specifies:

- Chained channel-program scheduling
- Printer or card punch
- ASA control character.

The module operates as follows:

- It receives control from a PUT routine when that routine finds that a buffer is to be scheduled, or from a WRITE routine at the conclusion of its processing.

- It adjusts the length entry and the start address entry in the channel program for either the control character or a variable-length block length field or for both, if both are present.

- It translates the control character and inserts it as the command byte of the Control CCW (which precedes the Write CCW).

- It attempts to join the current channel program to the preceding one (that is, chain schedule) by:

  Setting the ICB to not-complete.

  Inserting the address of the current channel program into the NOP CCW of the preceding channel program.

  Changing the NOP CCW in the preceding channel program to a TIC CCW.

  Updating the SAM IOB prefix block to point to the end of the current channel program.

- It determines whether the joining was successful by testing the ECB pointed to by the IOB for a completion posting by the I/O supervisor.

- If the I/O supervisor did not post the event as completed, the joining was successful and the routine returns control to the calling routine.

- If the I/O supervisor did post the event as completed, the routine tests the ICB for the present channel program to find whether the joining was successful or not.

- If the present ICB remains unposted, the present channel program was not joined to the preceding one. The routine prepares to cause restart of the channel by copying the channel program start address from the current ICB into the IOB, and uses the EXCP macro-instruction to cause scheduling of the channel program. It then returns control to the calling routine.

- If the present ICB is posted complete, the present channel program was joined successfully. (The routine was interrupted long enough, between the joining and the testing, for the channel program to be executed and for the channel end appendage to post the ICB.) The routine returns control to the calling routine.

TRACK OVERFLOW END-OF-BLOCK ROUTINE

The track overflow end-of-block routine processes channel programs for output data sets whose blocks may overflow from one track onto another. (See Figure 7.) Such a block is written by a channel program consisting of a channel program segment for each track to be occupied by a segment of the block. The track overflow end-of-block routine computes the address of each track written on; to progress from track to track (to continue writing successive segments of one block) the channel program uses the Search command with the multiple-track (M/T) mode.

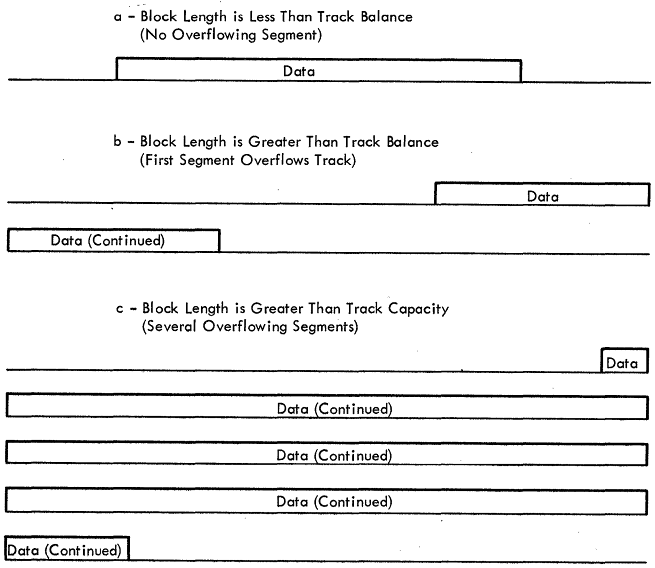

a - Block Length is Less Than Track Balance
(No Overflowing Segment)
Data
b - Block Length is Greater Than Track Balance
(First Segment Overflows Track)
Data
Data (Continued)
c - Block Length is Greater Than Track Capacity
(Several Overflowing Segments)
Data
Data (Continued)
Data (Continued)
Data (Continued)
Data (Continued)

Figure 7. Track Overflow Records

There is one track overflow end-of-block routine (module IGG019C2); it is used with output data sets. If the access conditions shown in Table 10 are specified for a DCB, the OPEN executor selects this routine, loads it, and places its address into the DCBEOB field or DCBEOBW. (For an input data set with track overflow, end-of-block module IGG019CC is used.)

Table 10. Module Selector - Track Overflow End-of-Block Module

Access Conditions	Selections
Output, Inout, Outin	X
Track Overflow	X
End-of-Block Module	
IGG019C2	X

End-of-Block Module IGG019C2

Module IGG019C2 performs device-oriented processing when track overflow is permitted with an output data set. The OPEN executor selects and loads this module If the OPEN parameter list specifies:

- Output, Inout, or Outin

and the DCB specifies:

- Track overflow.

The module operates as follows:

- It receives control from a PUT routine when that routine finds that a buffer is to be scheduled, or from a WRITE routine at the conclusion of its processing.
- It compares the block length with the space remaining on the track last written on.
- If the entire block fits on this track, the module completes a channel program (consisting of one channel program segment) for writing the block, updates the track balance, and passes control to the I/O supervisor.
- If at least a one-byte data-field fits on this track, the module completes a channel program segment for the segment of the block that fits on the track (by entering the Seek address, main storage address, and count field for the channel program segment) and tests if there is another track in the same extent.
- If the next track is in this extent, it compares the remaining block length with the track capacity.
- If the remainder of the block exceeds the track capacity, the module proceeds as when at least one byte fits on the track.
- If the remainder of the block is less than the track capacity, the module completes the final channel program segment for the final segment of the block, updates the track balance, and passes control to the I/O supervisor.
- If the next track is not in this extent, the module passes control to the track balance routine via an SVC 25 instruction. (That routine will erase all tracks in the current extent that were found insufficient for the block to be written.) On return of control from the track balance routine, the module tests if there is another extent.
- If there is another allocated extent on this volume, the module reconstructs the channel program by proceeding when at least one byte fits on a track.
- If there is no other allocated extent on this volume, an end-of-volume condition exists. The module sets the DCBCIND1 field in the DCB and the CSW field in the IOB to show this, and returns control to the PUT or WRITE routine without issuing an EXCP macro-instruction. The EOV condition is eventually recognized and processed, in QSAM by the synchronizing routine, in BSAM by the CHECK routine.

SYNCHRONIZING AND ERROR PROCESSING ROUTINES

A synchronizing and error processing routine synchronizes execution of the processing program with execution of the channel programs, and performs error processing to permit continued access to the data set after an error was encountered during the execution of a channel program.

There are five synchronizing and error processing routines. Four of the five routines:

- Are unique to QSAM
- Both synchronize and process errors
- Receive control from a GET or a PUT routine
- Are pointed to by an address in the DCB.

The fifth routine, the track overflow asynchronous error processing routine:

- Is shared between QSAM and BSAM
- Only processes errors
- Receives control by being scheduled by the track overflow abnormal end appendage
- Is pointed to by an address in an interruption request block (IRB).

To synchronize, the QSAM Input and Output Synchronizing and Error Processing routines (modules IGG019AQ and IGG019AR) return control to the GET or PUT routine immediately if the channel program executed without error; or use the WAIT macro-instruction if the channel program has not yet executed. To process errors, these routines pass control to the SYNAD/EOV executor (using SVC 55) to distinguish between the processing necessary for unit check - that is, a permanent error - and unit exception - that is, an end-of-volume condition.

For a unit check the executor returns control to the synchronizing routine, which in turn passes control to the SYNAD routine. On return of control from the SYNAD routine, the synchronizing routine again passes control to the executor to implement the error options. For the ACCEPT and SKIP options, control returns once more to the synchronizing routine. It now operates as when it is first entered.

For a unit exception the executor causes end-of-volume processing by the end-of-volume routine of I/O support. That routine passes control to the EOV/new volume executor. The executor returns control to the synchronizing routine. It now operates as when it is first entered.

To synchronize the Paper Tape Character Conversion Synchronizing routine (contained in the paper tape GET module IGG019AT) returns control to the GET routine immediately if the channel program executed without error; or uses the WAIT macro-instruction if the channel program has not yet executed. To process errors, the routine passes control to the SYNAD routine. When control returns from the SYNAD routine to the synchronizing routine, the latter implements the error option. (The equivalent of an end-of-volume condition is handled by the paper tape GET routine.)

To synchronize, the Update Synchronizing and Error Processing routine (module IGG019AF), returns control to the GET routine immediately if the channel program executed without error; or uses the WAIT macro-instruction if the channel program has not yet executed. To process an end-of-volume condition, the routine suspends volume-switching until processing on the old volume is finished. To process permanent errors, the routine interprets the error option to assure that neither a buffer nor a block is skipped.

The error processing performed by the track overflow asynchronous error processing routine (module IGG019C1) distinguishes two kinds of errors - those in the block being read and those in the block being skipped over to read the next one. For errors in the block being read, the routine sets the channel program to permit the processing program to continue reading the segments and blocks beyond the one in error; for errors in the block being skipped, the routine resets the channel program and uses the EXCP macro-instruction, so that the processing program is unaware of the error.

For an error whose character and occurrence the processing program must know about (errors in segments of the block being read into the buffer), the track overflow routine addresses the IOB to the next track and its channel program and causes control to return to the processing program via the TCB queue. For errors whose correction does not affect the processing program (errors in segments of the block being skipped over), the module uses the EXCP macro-instruction to skip around the defective segment to present the processing program with the block it expects to obtain. This latter condition only holds if an error occurs on a Read-Data CCW with the Skip bit on for a segment that is not the last or only segment on an alternate track. In that case control returns to the processing program when the desired block is in the buffer in its entirety. For errors that do not permit reading the entire block in one pass without error, control returns to the processing program with the IOB set to a track and channel program that permits reading the segments following the defective one. The defective segment and the preceding good segments of the block are in the buffer at the time control is returned to the processing program.

Four of the five routines described here (those enumerated in Table 11) are unique to QSAM. One of these routines gains control when a GET or a PUT routine finds that it needs a new buffer. Table 11 lists the routines available and the conditions that cause a particular routine to be used. The OPEN executor selects one of the routines, loads it, and puts its address into the DCBGERR/PERR field.

The fifth routine (identified in Table 12) is shared between QSAM and BSAM. It gains control be being scheduled for eventual execution by track overflow flow abnormal end appendage IGG019C3. The OPEN executor loads it and enters its address in an IRB; the address of the IRB is in the DEB. (If QSAM is used, module IGG019AQ is also used.)

Table 11. Module Selector - Synchronizing and Error Processing Modules

Access Conditions	Selections			
GET	X	X		X
PUT			X	
Input, Readback		X		
Output			X	
Update	X			
Paper tape character conversion				X
Modules				
IGG019AF	X			
IGG019AQ		X		
IGG019AR			X	
IGG019AT[1]				X

[1] This module includes both the paper tape synchronizing and error processing routine and the paper tape GET routine. Both routines are described in the GET routines section of this publication.

Table 12. Module Selector - Track Overflow Asynchronous Error Processing Module

Access Conditions	Selections	
GET	X	
READ		X
Input, Inout, Outin	X	X
Track Overflow	X	X
Module		
IGG019C1	X	X

SYNCHRONIZING MODULE IGG019AF (UPDATE)

Module IGG019AF finds the next buffer and assures that it has been refilled. If a unit status prevented refilling the buffer, the module processes the pending channel programs according to whether they are Empty-and-Refill or Refill-only channel programs. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Update

and the DCB specifies:

- GET.

The module operates as follows if no error occurred:

- It receives control when the Update GET routine finds that a new buffer is needed. It also receives control after the FEOV macro-instruction is encountered in a processing program, once from the Update GET routine (when the FEOV executor schedules the last buffer) and once directly from the FEOV executor (when it awaits execution of the scheduled buffers.)
- If the next buffer has been refilled, the module returns control to the Update GET routine.
- If the channel program for the next buffer has not yet executed, the module awaits its execution.

The module operates as follows if an end-of-volume condition was encountered:

- It receives control when the Update GET routine finds that a new buffer is needed or when the FEOV executor awaits execution of the scheduled buffers.
- If the channel program for the next buffer encountered an end-of-volume condition, or if control has come to this module due to an FEOV macro-instruction, the module finds the IOBs flagged for output. It then resets the command-chain flag at the end of the Empty portion of the channel program to off, and schedules the Empty channel programs for execution via an EXCP macro-instruction.
- If all Empty channel programs have been executed, or if none are pending, the module passes control to the SYNAD/EOV executor via an SVC 55 instruction. If this module has control due to an FEOV macro-instruction, control returns to the routine that passed control.

- If a permanent error is encountered during execution of Empty channel programs for an end-of-volume condition or for an FEOV macro-instruction, control passes to the SYNAD routine, if one is present. The SYNAD routine returns control to this module.

- The module then processes the error option as follows:

  - Accept or Skip Option:

    The pending Empty channel programs are rescheduled for execution via EXCP macro-instructions.

  - Terminate Option:

    Control passes to the ABEND routine.

The module operates as follows if a permanent error was encountered:

- It receives control when the Update GET routine finds that a new buffer is needed.

- If the channel program for the next buffer encountered a permanent error and a SYNAD routine is present, the module passes control to the SYNAD routine.

- If control returns from the SYNAD routine, or if there is no SYNAD routine, the module processes the error option in the following manner:

  - Accept Option:

    If the error occurred in the Empty portion of a channel program, the module resets the IOB to point to the Refill portion of the channel program and issues an EXCP macro-instruction for it and all following IOBs.

    If the error occurred in the Refill portion of a channel program, the module posts the current IOB as complete without error and issues an EXCP macro-instruction for all the IOBs except the present one.

    The module assures refilling of the buffer associated with the first IOB and then returns control to the Update GET routine.

  - Skip Option:

    If the error occurred in the Empty portion of a channel program, the module operates as it does for the Accept option.

    If the error occurred in the Refill portion of a channel program, the module issues an EXCP macro-instruction for all IOBs.

    The module assures refilling of the buffer associated with the first IOB and then returns control to the Update GET routine.

  - Terminate Option:

    If the error occurred in the Empty portion of a channel program, the module passes control to the ABEND routine.

    If the error occurred in the Refill portion of a channel program, the module finds the end of the Empty portion of any pending Empty-and-Refill channel programs, resets the command-chain flag to off, and issues an EXCP macro-instruction for these Empty channel programs. On execution of all the channel programs, the module passes control to the ABEND routine.

SYNCHRONIZING MODULE IGG019AQ (INPUT)

Module IGG019AQ finds the next input buffer, determines its status, and passes a full buffer to the GET routine. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Input, Readback

and the DCB specifies:

- GET.

The module operates as follows:

- It receives control when a GET routine determines that a new buffer is needed.

- It finds the next IOB and tests the status of the channel program associated with that IOB.

- If the channel program is not yet executed, the module issues a WAIT macro-instruction.

- If the channel program has been executed normally, the module updates the DCBIOBA field to point to this IOB and returns control to the GET routine.

- If an error occurred during the execution of the channel program, the module issues an SVC 55 instruction to pass control to the SYNAD/EOV executor (IGC0005E). (For an EOV condition, control eventually passes to the end-of-volume routine of I/O support and returns after the next volume has been found and the purged channel programs have been rescheduled. For a description of the flow of control from the SYNAD/EOV executor for a permanent error condition, refer to the section: Sequential Access Method Executors, in this publication.)

SYNCHRONIZING MODULE IGG019AR (OUTPUT)

Module IGG019AR finds the next output buffer, determines its status, and passes an empty buffer to the PUT routine. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Output

and the DCB specifies:

- PUT.

The module operates as follows:

- It receives control when a PUT routine determines that a new buffer is needed.

- It finds the next IOB and tests the status of the channel program associated with that IOB.

- If the channel program is not yet executed, the module issues a WAIT macro-instruction.

- If the channel program has been executed normally, the module updates the DCBIOBA field to point to this IOB and returns control to the PUT routine.

- If an error occurred during the execution of the channel program, the module issues an SVC 55 instruction to pass control to the SYNAD/EOV executor (IGC0005E). (For an EOV condition, control eventually passes to the end-of-volume routine of I/O support and returns after a new volume or more space has been found and the purged channel programs have been rescheduled. For a description of the flow of control from the SYNAD/EOV executor for a permanent error condition, refer to the section: Sequential Access Method Executors, in this publication.)

TRACK OVERFLOW ASYNCHRONOUS ERROR PROCESSING MODULE IGG019C1

Module IGG019C1 (used in both QSAM and BSAM) processes error conditions that are encountered in the execution of a channel program for an input data set with track overflow. Its processing of error conditions is asynchronous to the execution of the channel program, the I/O supervisor, or the processing program. It receives control by being scheduled for execution by the track overflow abnormal end appendage IGG019C3. It passes control to the processing program through the supervisor. The module determines the Seek address for reading the segments and blocks beyond the segment in error and inserts it in the IOBSEEK field. If the error occurred in a segment of the block being read into the buffer, the segment following the segment in error is read, if the processing program chooses the ACCEPT option in the SYNAD routine. If the error occurred in a segment in the block preceding the block to be read into the buffer (that is, the error occurred in the block being skipped over to find the block to be read into the buffer), the wanted block is in the buffer when the processing program obtains the buffer.

The OPEN executor selects and loads this module (and places its address into an IRB pointed to in the DEB) if the OPEN parameter list specifies:

- Input, Inout, or Outin

and the DCB specifies:

- Track Overflow
- GET or READ.

The module operates as follows if the error occurred in a CCW other than a Read-Data CCW:

- It receives control from the supervisor.

- It increases the track address in the IOB by 1, posts the ECB with the error code, and causes control to return to the processing program.

The module operates as follows if the error occurred in a Read-Data CCW (without a Skip bit on):

- It receives control from the supervisor.

- If the segment in error is the last or only segment of the block, the module posts the ECB with the error code and causes control to return to the processing program.

- If the segment in error is not the last segment and it is not on an alternate track, the module sets the IOB to address the track following the track in error, posts the ECB with the error code, and causes control to return to the processing program.

- If the segment in error is not the last segment and it is on an alternate track, the module increases the track address in the IOB by 1, posts the ECB with the error code, and causes control to return to the processing program.

The module operates as follows if the error occurred in a Read-Data CCW with the Skip bit on:

- It receives control from the supervisor.

- If the segment in error is the final or only segment of a block and it is not on an alternate track, the module sets the IOB to address the track in error, changes the Read-Data command to a NOP command and issues an EXCP macro-instruction for the changed channel program.

- If the segment in error is the final or only segment of a block and it is on an alternate track, the module sets the IOB to address the track following the one originally addressed, posts the ECB with the error code, and causes control to return to the processing program. (In the case of an error in a final or only segment on an alternate track, the remaining segment or blocks on that track will not be read.)

- If the segment in error is not the last one and it is not on an alternate track, the module sets the IOB to address the track following the one in error and issues an EXCP macro-instruction for the readdressed channel program.

- If the segment in error is not the last one and it is on an alternate track, the module successively increases the track address in the IOB by 1 and issues an EXCP macro-instruction for the readdressed channel program.

- When control returns from the I/O supervisor, this module awaits execution of the channel program via a WAIT macro-instruction. On channel program execution, the module restores the purged IOBs (and the Read-Skip command, if it was changed to a NOP command) and causes control to return to the processing program.

APPENDAGES

Appendages are access method routines that receive control from and return control to the I/O supervisor and that operate in the supervisor state. (The same appendages used in QSAM and in BSAM.) An appendage that receives control from the I/O interruption supervisor, tests and may alter the channel status word (CSW). The I/O interruption supervisor uses the CSW to post the event control block (ECB). An appendage that receives control from the EXCP supervisor, before the latter causes execution of the channel program by using the SIO instruction, may update or alter channel commands just before channel program execution. The relationship of the I/O supervisor and the appendages are illustrated in Figure 8.

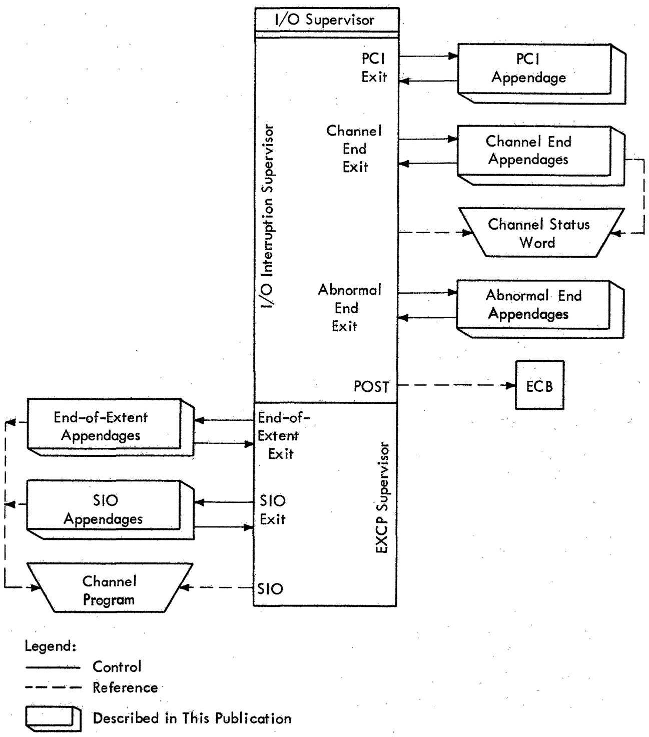


Figure 8. Relationship of I/O Supervisor and Appendages

The I/O supervisor permits an appendage to gain control at certain exit points. At that time the I/O supervisor refers to the entry associated with that exit in the appendage vector table (whose address is in the data extent block - DEB). If an entry contains the address of an appendage, control passes to it; else control remains with the I/O supervisor. The five I/O supervisor exits, at which appendages receive control, are:

- End-of-Extent
- SIO
- Channel End
- PCI
- Abnormal End.

Appendages differ from other sequential access method routines that are loaded by the OPEN executor into processing program main storage in that they operate in the supervisor state and in that they operate asynchronously with the processing program, that is, the events that cause them to gain control depend, not on the progress of the processing program, but on the progress of the channel program. There are twelve appendages. Nc, one, or several appendages may be used with one DCB. Table 13 lists the appendages, the conditions that cause the different appendages to be used, and the I/O supervisor exits that pass control to them. The OPEN executor selects and loads all the necessary appendages to be used with that DCB, and places their addresses into the various fields of the appendage vector table. For example, if the Update mode of OPEN is specified, appendage IGG019CG, associated with the SIO appendage exit, is selected and loaded by the Open executor.

END-OF-EXTENT APPENDAGES

End-of-extent appendages gain CPU control if the EXCP supervisor finds an end-of-extent condition. This condition exists if the direct-access device storage address associated with a channel program is outside of the extent currently pointed to in the data extent block (DEB).

Four end-of-extent appendages are provided for use with sequential access method routines:

- IGG019AW processes an end-of-extent condition for QSAM Update mode channel programs.
- IGG019BM processes an end-of-extent condition for BSAM Update mode channel programs.
- IGG019CH processes an end-of-extent condition when neither the Update mode nor chained channel-program scheduling is specified.
- IGG019CZ processes end-of-extent conditions when chained channel-program scheduling is used.

Appendage IGG019AW (End-of-Extent - Update - QSAM)

Appendage IGG019AW readdresses the Refill portions of all QSAM Update channel programs to a new extent. The OPEN executor selects and loads this module for use as the end-of-extent appendage if the OPEN parameter list specifies:

- Update

and the DCB specifies:

- GET.

The appendage operates as follows:

- It receives control from the EXCP supervisor under one of the following conditions:

  A Refill portion of QSAM Update channel program attempts to read the first block beyond the present extent.

  The remaining channel programs attempt to refill their buffers from the new extent.

- If there is no other extent, the appendage sets error indications in the IOB and the DCB (to show an end-of-volume condition) and returns control to the EXCP supervisor. The EXCP supervisor then issues a PURGE macro-instruction for that channel program. (The Update synchronizing routine assures writing out of the Empty portions of pending channel programs.)
- If the interruption occurred in a Read-Count CCW and there is a new extent, the appendage builds a Seek address for the new extent using the starting address from the DEB. It then copies this new Seek address into the IOB and UCB (unit control block), and updates the M value in the Refill portion of each channel program.
- If the interruption occurred in a Seek CCW, the appendage copies the Seek address from the Refill portion of the present channel program into the IOB and UCB.
- It resets the IOB and UCB to address the next track and its channel program and returns control to the I/O supervisor.

Table 13. Module Selector - Appendages

Access Conditions	Selections											
Input, Inout, Outin				X	X	X						
Readback					X							
Update	X	X	X									
Sysin							X					
GET	X											
READ		X										
Record format is fixed-length									X			
Record format is fixed-length blocked					X							
Record format is variable-length						X						
Record format is not fixed-length standard				X								
Direct-access storage				X							X	
Printer								X				
Paper tape									X			
Chained scheduling										X	X	
Track overflow												X
Appendages												
IGG019AW	AW											
IGG019BM		BM										
IGG019CG			CG									
IGG019CH				CH								
IGG019CI					CI							
IGG019CJ						CJ						
IGG019CK							CK					
IGG019CL								CL				
IGG019CS									CS			
IGG019CU										CU		
IGG019CZ											CZ	
IGG019C3												C3
Exits												
End-of-Extent	AW	BM		CH							CZ	
SIO			CG					CL				
Channel End					CI	CJ	CK		CS	CU		
PCI										CU		
Abnormal End										CU		C3

Appendage IGG019BM (End-of-Extent - Update - BSAM)

Appendage IGG019BM readdresses channel programs to a new extent for a DCB opened for Update and using BSAM. The OPEN executor selects and loads this appendage for use as the end-of-extent appendage if the OPEN parameter list specifies:

- Update

and the DCB specifies:

- READ.

The appendage operates as follows:

- It receives control from the EXCP supervisor when a channel program to refill a buffer attempts to read the first block beyond the present extent.
- If there is no other extent (for a Refill channel program), the appendage sets error indications in the IOB and the DCB (to show an end-of-volume condition) and returns control to the EXCP supervisor.
- If there is a new extent (for a Refill channel program), the appendage adds 1 to the value of M in the DCBFDAD field and in the Seek address of each Refill channel program for the DCB. It places the new Seek address into the current IOB and into the UCB, and returns control to the EXCP supervisor. The supervisor restarts the channel program.

Appendage IGG019CH (End-Of-Extent - Ordinary)

Appendage IGG019CH finds a new extent when the EXCP supervisor finds an end-of-extent extent condition. The OPEN executor selects and loads this appendage for use as the end-of-extent appendage if the OPEN parameter list specifies:

- Input, Inout, or Outin

and the DCB specifies:

- Direct-access storage device
- Record format other than fixed-length standard
- Normal channel-program scheduling.

The appendage operates as follows:

- It receives control when a channel program attempts to read a block beyond the present extent.
- The appendage examines the DEB for another extent.
- If there is another extent, the appendage enters the new full device address in the DCB, the unit control block (UCB), and the IOBs, and returns control to the EXCP supervisor. The EXCP supervisor restarts the channel program.
- If there is no other extent, the appendage sets error indications in the IOB and the DCB (to show an end-of-volume condition) and returns control to the EXCP supervisor. The EXCP supervisor then issues a PURGE macro-instruction for that channel program.

Appendage IGG019CZ (End-of-Fxtent - Chained Channel-Program Scheduling)

Appendage IGG019CZ readdresses the chain of channel programs to a new extent when the EXCP supervisor finds an end-of-extent condition. The OPEN executor selects and loads this appendage for use as the end-of-extent appendage if the DCB specifies:

- Chained channel-program scheduling
- Direct-access storage device.

The appendage operates as follows:

- It receives control when an end-of-track condition interrupts the chained scheduling and the I/O supervisor finds that the next track is not in the current extent.
- If there is another extent, the appendage enters the new Seek address in the DCB, IOB, and unit control block (UCB), updates the Seek addresses of the remaining ICBs, and returns control to the I/O supervisor to reschedule the channel program for execution.
- If there is no other extent, the appendage sets a volume-full indication in the DCB, IOB, and ICB and returns control to the I/O supervisor to skip further scheduling for this DCB.

START I/O (SIO) APPENDAGES

Start I/O (SIO) appendages, if present, gain CPU control when the start I/O subroutine of the EXCP supervisor reaches the start I/O appendage exit. These appendages set channel program entries whose value depends on events associated with the execution of the preceding channel program. There are two SIO appendages:

- IGG019CG. This appendage makes the Seek address accessible to the I/O supervisor for QSAM and BSAM Update channel programs that refill buffers. (This is necessary because the Seek address for such a channel program is read in by the preceding channel program into a location unknown to the I/O supervisor.)

- IGG019CL. This appendage causes the next line to print at the top of a new page if a printer overflow condition was encountered in the execution of the last channel program.

Appendage IGG019CG (SIO - Update)

Appendage IGG019CG resets the IOB to the Seek address and channel program for refilling for a Refill-only Update channel program. The OPEN executor selects and loads this appendage for use as the SIO appendage if the OPEN parameter list specifies:

- Update.

The appendage operates as follows:

- It receives control whenever the EXCP supervisor reaches the SIO appendage exit.

- It tests the IOB to determine whether the buffer is to be emptied and refilled or to be refilled only.

- If the buffer is to be emptied and refilled, the module returns control to the EXCP supervisor.

- If the buffer is to be refilled only, the module resets the IOB to the Refill portion of the channel program and its Seek address and returns control to the EXCP supervisor.

Appendage IGG019CL (SIO - PRTOV)

Appendage IGG019CL causes a skip to the top of a new page with the first channel program following a printer overflow condition. The OPEN executor selects and loads this appendage for use as the SIO appendage if the DCB specifies:

- Printer.

The appendage operates as follows:

- The appendage tests the IOB to determine whether a PRTOV macro-instruction was issued with this PUT or WRITE macro-instruction.

- If a PRTOV macro-instruction was not issued, the appendage returns control to the EXCP supervisor immediately.

- If the PRTOV macro-instruction was issued, the appendage resets the PRTOV bit in the IOB and tests the DCBIFLGS field to determine whether a printer overflow condition has occurred.

- If printer overflow has not occurred, the appendage returns control to the EXCP supervisor.

- If printer overflow has occurred, the appendage resets the DCBIFLGS field, inserts the "skip to 1" command byte into the channel program, updates the IOB channel program start address field and the channel address word (location 72), and returns control to the EXCP supervisor.

CHANNEL END APPENDAGES

Channel end appendages, if present, gain CPU control when the I/O interruption supervisor reaches the channel end appendage exit. For a SYSIN data set, the SYSIN appendage recognizes the delimiter characters. For other data sets, other appendages distinguish between valid and invalid block lengths by computation. The five channel end appendages are:

- IGG019CI. This appendage distinguishes between wrong-length and truncated blocks when fixed-length blocked records are being read using normal channel program scheduling.

- IGG019CJ. This appendage distinguishes between wrong-length and variable-length blocks when variable-length records are being read using normal channel program scheduling.

- IGG019CK. This appendage recognizes SYSIN delimiter characters.

- IGG019CS. This appendage distinguishes between valid and invalid wrong-length indications when paper tape is being read.

- IGG019CU. This appendage (which also appears at the PCI and abnormal end exits), parts executed channel programs that were scheduled by chaining, and posts the completions. For channel end channel status, this appendage distinguishes between wrong-length and truncated blocks when fixed-length blocked records are being read using chained channel-program scheduling.

(Refer to the section for PCI appendages for a discussion of parting of chained channel-programs and a description of appendage IGG019CU.)

Appendage IGG019CI (Channel End - Fixed-Length Blocked Record Format)

Appendage IGG019CI distinguishes between valid wrong-length blocks and truncated blocks. The OPEN executor selects and loads this appendage if the OPEN parameter list specifies:

- Input, Readback, Inout, or Outin

and the DCB specifies:

- Fixed-length blocked records.

(Under these conditions the SLI flag is off in the Read channel command word.)

The appendage operates as follows:

- It receives control when the I/O interruption supervisor arrives at the channel end exit.

- If the appendage finds either the unit exception bit on in the channel status word, or the wrong-length indication off, it returns control to the I/O interruption supervisor immediately.

- The appendage calculates the length of the block and compares this length to that in the DCBLRECL field.

- If the fixed-length blocked record format is specified and the block length is an integral multiple of the DCBLRECL field value (showing it to be a truncated block), the appendage turns off error indications in the ECB and the DCB and returns control to the I/O interruption supervisor.

- If the fixed-length blocked standard record format is specified and the block is a truncated block, the appendage determines that this is the last block of the data set. The appendage sets bits in the DCB and the ECB to show that an end-of-volume (EOV) condition exists, and returns control to the I/O interruption supervisor.

- If the block length is not an integral multiple, the appendage returns control to the I/O interruption supervisor immediately. The I/O interruption supervisor then sets the ECB to show that the channel program was executed with an error condition.

Appendage IGG019CJ (Channel End - Variable-Length Record Format)

Appendage IGG019CJ distinguishes between valid wrong-length blocks and variable-length blocks. The OPEN executor selects and loads this appendage if the OPEN parameter list specifies:

- Input, Inout, Outin

and the DCB specifies:

- Variable-length records.

(Under these conditions the SLI flag is off in the Read channel command word.)

The appendage operates as follows:

- It receives control when the I/O interruption supervisor arrives at the channel end exit.

- If the appendage finds a unit exception bit on in the channel status word, it returns control to the I/O interruption supervisor immediately.

- The appendage calculates the length of the block and compares it to that in the block length field.

- If the lengths are equal, the appendage turns off error indications in the ECB and DCB and returns control to I/O interruption supervisor.

- If the lengths are not equal, control is returned to the I/O interruption supervisor immediately. The I/O interruption supervisor then sets the ECB to show that the channel program executed with an error condition.

Appendage IGG019CK (Channel End - SYSIN)

Appendage IGG019CK translates the delimiter character for a SYSIN data set into an end-of-data-set indication for the access method routine. The OPEN executor selects and loads this appendage if the device assigned to this DCB is SYSIN.

The appendage operates as follows:

- It receives control when the I/O interruption supervisor arrives at the channel end exit.
- The appendage tests the buffer for the SYSIN delimiter characters /*.
- If the characters read are not delimiter characters, the appendage returns control to the I/O supervisor.
- If the characters read are delimiter characters, the appendage turns on the unit exception bit in the channel status word and the error flag in the DCB, indicating an end-of-data set condition, and returns control to the I/O supervisor.

Appendage IGG019CS (Channel End - Paper Tape)

Appendage IGG019CS distinguishes between valid wrong-length blocks and the wrong-length indication characteristic when paper tape is being read. The OPEN executor selects and loads this appendage if the DCB specifies:

- Fixed-length record format
- Paper Tape.

The appendage operates as follows:

- It receives control when the I/O interruption supervisor arrives at the channel end exit.
- If the channel status word (CSW) residual count is zero, the appendage turns off error indications in the IOB and the DCB and then returns control to the I/O supervisor.
- If the channel status word (CSW) residual count is not zero, the appendage returns control to the I/O supervisor immediately.

PROGRAM CONTROLLED INTERRUPTION (PCI) APPENDAGE (EXECUTION OF CHANNEL PROGRAMS SCHEDULED BY CHAINING)

There is one program controlled interruption (PCI) appendage. If chained channel-program scheduling is used, its address is placed into the appendage vector table for all three I/O interruption supervisor exits: PCI, channel end, abnormal end.

A program controlled interruption (PCI), in the sequential access methods, signals the normal execution of a channel program that was scheduled by chaining. The interruption occurs when control of the channel has passed to the next channel program. If the only channel status is PCI the I/O supervisor performs no processing; if other channel conditions are also present, the I/O supervisor processes these in normal fashion after it regains CPU control from the PCI appendage.

This appendage performs the following three functions:

- It performs the channel status analysis usually done by the I/O interruption supervisor. The interruption is caused by a condition in the logic of the channel program rather than a condition in the channel or the device. The condition is meaningful only to the processing program (in this case, the access method routines, or, more specifically, the appendage) and has no meaning to the I/O supervisor.
- It repeats this process for preceding channel programs whose PCIs were lost. PCIs are not stacked. If a channel remains masked from the time of one PCI until after another PCI, only one PCI occurs.
- It performs processing normally necessary for other interruptions (for example, channel end). Interruptions other than PCIs may terminate execution of chained channel programs.

Accordingly, a PCI appendage not only does the processing implicit for the logical condition that the interruption signals (namely, that the preceding channel program executed normally), but also extends this processing back to any preceding channel programs whose PCI may have been masked and, finally, takes CPU control at other I/O interruption supervisor appendage exits if chained channel-program scheduling is used.

Appendage IGG019CU (Channel End, PCI, Abnormal End - Chained Channel-Program Execution)

Appendage IGG019CU disconnects (parts) chained channel programs that have executed and posts their completion; in addition, it performs normal channel end and abnormal end appendage processing. (For a description of the joining process of chained channel-program scheduling refer to the descriptions of the chained channel-program scheduling end-of-block routines.) The OPEN executor selects and loads this appendage for use as the channel end, PCI, and abnormal end appendage if the DCB specifies:

- Chained channel-program scheduling.

The appendage operates as follows:

- It receives control from the I/O interruption supervisor when the latter arrives at the PCI, channel end, and abnormal end appendage exits.

- It tests whether the CSW and the IOB field "First ICB" point to the same channel program.

- If they do, the appendage returns control to I/O supervisor, unless a channel end condition exists.

- If they do not, the appendage disconnects (parts) the channel program (pointed to by the ICB) from the next channel program in the chain as follows:

  For input, the appendage tests the IOB for an end-of-volume condition. If it exists, the appendage continues as for a channel end interruption with a permanent error.

  For output, or for input without an associated end-of-volume condition, the appendage resets the command in the last CCW from TIC to NOP and the address to the beginning of the next channel program.

  If the device is magnetic tape, it updates the DCBBLKCT field in the DCB.

  If a WAIT macro-instruction was addressed to this channel program, the appendage causes the POST routine to perform its processing and to return control to the appendage.

  It posts the ICB with the completion code and with channel end and updates the IOB SAM prefix to point to the next ICB.

  It repeats this parting process until the IOB and the CSW point to the same channel program.

The appendage continues as follows if channel end occurred without an error:

- It sets the IOB and the ICB to show the channel program completed without error, and resets the IOB to point to the next channel program and ICB.

- If there are more channel programs to be executed, the appendage resets the IOB to not-complete and passes control to the EXCP supervisor tc schedule these channel programs.

- If there are no more channel programs to be executed, the appendage returns control to the I/O supervisor for normal

The appendage continues as follows if the channel end interruption occurred with a wrong length indication:

- It determines whether a truncated block has been read.

- If a truncated block has been read in a data set with fixed-length blocked standard record format, it sets:

  the DCB to show an end-of-volume condition,

  the current ICB to complete-without-error.

  the next ICB to complete-with-error,

  the CSW in the next ICB to show channel end and unit exception.

  It returns control to the I/O interruption supervisor.

- If a truncated block has been read in a data set with fixed-length blocked record format, the appendage sets the ICB to complete-without-error and resets the IOB to point to the next ICB and its channel program. The appendage causes control to pass to the EXCP supervisor to restart the channel.

- If a block with wrong length data has been read, the appendage continues as for permanent errors.

The appendage continues as follows if channel end occurred with an error:

- It isolates the channel program in-error by parting it from the next one.
- It sets the IOB to point to the channel-program in-error.
- It sets the DCB to show that the channel program is being retried.
- It returns control to the I/O interruption supervisor. That routine then processes the channel program in the Error Retry procedure.

The appendage continues as follows if channel end occurred with a permanent error:

- It receives control after the I/O supervisor Error Retry procedure is found unsuccessful in correcting the error.
- It posts the ICB to show that the channel program as completed in-error.
- It parts the channel program in-error from the following one.
- It resets the IOB to point to the channel program after the one in-error.
- It returns control to the I/O interruption supervisor.

ABNORMAL END APPENDAGES

Abnormal end appendages receive control from the I/O interruption supervisor when the latter finds a unit check condition in the channel status word (CSW). The appendages for this exit are a track overflow appendage and a chained channel-program execution appendage shared with the channel end and PCI exits. The shared appendage is described under the PCI appendage.

A unit check status in a channel addressing an input data set with track overflow may indicate a permanent error in one segment of a block. If there are further good segments, or if the segment in error is being skipped over to find the next block, the sequential access methods attempt to continue access beyond the segment in error. The processing necessary to accomplish this is performed by the track overflow asynchronous error processing routine (module IGG019C1, described in the synchronizing and error processing routines section), rather than by the appendage. To permit other I/O operations to continue, the appendage suspends further processing of the condition by the I/O supervisor, schedules the asynchronous error processing routine, and returns control to the I/O supervisor.

Appendage IGG019C3 (Abnormal End - Track Overflow)

Appendage IGG019C3 schedules the track overflow asynchronous error processing routine if a permanent error occurs in a channel program for an input data set with track overflow. The OPEN executor select and loads this appendage for use as the abnormal end appendage if the OPEN parameter list specifies:

- Input, Inout or Outin

and the DCB specifies:

- Track overflow.

The appendage operates as follows:

- It receives control from the I/O interruption supervisor when the latter reaches the abnormal end appendage exit.
- If the CSW that caused this appendage to gain control addresses a Read-Data CCW (without a Skip bit) and shows a unit exception channel status, the appendage returns control to the I/O interruption supervisor without further processing. (After control returns to the processing program, the synchronizing or CHECK routine processes this channel status as an end-of-volume condition.)
- If the CSW that caused this appendage to gain control addresses a Read-Data CCW (with a Skip bit on) and shows a unit exception or a unit check channel status, the appendage passes control to the exit effector routine together with the entry point address of I/O supervisor that causes the I/O supervisor not to post the ECB and to retain the request element for the channel program. (The exit effector routine will schedule the track overflow asynchronous error processing routine for eventual execution and pass control to the given entry point.)

QSAM CONTROL ROUTINES

These control routines, shared by QSAM and BSAM, consist of both modules loaded by the OPEN executor and macro-expansions. The selection and loading of one of the modules is done by the OPEN executor and depend on the access conditions; the presence of macro-expansions depends solely on the use of the corresponding macro-instruction in the processing program and is independent of the presence or absence of modules.

If a CNTRL macro-instruction is encountered in a processing program using QSAM or BSAM, control passes to a control routine. The PRTOV macro-expansions place the code to be executed in-line in the processing program. CNTRL routines pass control to the I/O supervisor; the macro-expansions return control to the processing program. The CNTRL routine for the card reader causes execution of a channel program that stacks the card just read into the selected stacker. The CNTRL routine for the printer causes execution of a channel program with a command to space or to skip. The printer overflow macro-expansions cause the printer overflow condition to be sensed for.

There are two CNTRL routines in QSAM; they are load modules. Table 14 lists the routines available and the conditions that cause a particular routine to be used. The OPEN executor selects one of the modules, loads it, and puts its address into the DCBCNTRL field.

Table 14. Module Selector - Control Modules

Access Conditions	Selections	
CNTRL	X	X
Printer	X	
Card Reader, a single buffer		X
Modules		
IGG019CA	X	
IGG019CB		X

There are two PRTOV routines; they are macro-expansions. Whenever the assembler encounters either of the two macro-instructions shown in Table 15, it substitutes the corresponding macro-expansion in the processing program object module.

Table 15. Control Routines That Are Macro-Expansions

Macro-Instruction	Number of Macro-Expansions
PRTOV - User exit	1
PRTOV - No user exit	1

CONTROL MODULE IGG019CA (CNTRL - SELECT STACKER - CARD READER)

Module IGG019CA permits stacker selection on the card reader. The OPEN executor selects and loads this module if the DCB specifies:

- CNTRL
- Card reader
- One buffer.

The module operates as follows:

- It receives control when the CNTRL macro-instruction is encountered in a processing program.
- For QSAM, the module schedules a channel program which stacks the card just read, reads the next card into the buffer, and returns control to the processing program. (Card reader GET modules IGG019AG and IGG019AH depend on the use of this routine to refill empty buffers.)
- For BSAM, the module schedules a channel program which stacks the card just read, and then returns control to the processing program. (The READ/WRITE module IGG019BA causes a channel program to be scheduled that reads the next card into the buffer.)

CONTROL MODULE IGG019CB (CNTRL - SPACE, SKIP - PRINTER)

Module IGG019CB causes printer spacing and skipping by use of macro-instructions; the spacing or skipping to be performed are specified as operands of the macro-instruction. The OPEN executor selects and loads this module if the DCB specifies:

- CNTRL
- Printer.

The module constructs a channel program to control the device, issues an EXCP macro-instruction and then returns control to the processing program.

PRINTER OVERFLOW MACRO-EXPANSIONS

The PRTOV macro-expansions permit processing program response to printer overflow conditions.

The following macro-expansions are created as in-line coding during the expansion of the macro-instruction.

PRTOV - User Exit

The coding operates as follows:

- A WAIT macro-instruction is issued for the IOB pointed to by the DCBIOBA field.
- The DCBIFLGS field of the DCB is tested for an overflow condition.
- If an overflow condition exists, a BALR instruction is issued to pass control to the user's routine.
- If no overflow condition exists, control passes to the next instruction.

PRTOV - No User Exit

The coding creates a test mask in the DCB field located at (DCBDEVT+1) and returns control to the processing program.

(The printer end-of-block routine temporarily stores the mask in the NOP channel command word (CCW) preceding the Write CCW, turns on a bit in the first byte of the IOB and resets the mask. The PRTOV appendage tests the IOB bit, to determine whether to respond to, or ignore, an overflow condition, and resets it.)

Basic sequential access method (BSAM) routines cause storage and retrieval of blocks of data. BSAM routines furnish device control, but do not provide blocking. There are six types of BSAM routines:

- READ routines.
- WRITE routines.
- End-of-block routines.
- CHECK routines.
- Appendages.
- Control routines.

Figure 9 and Table 16 show the relationship of BSAM routines, other portions of the operating system, and the processing program.

A READ or a WRITE routine receives control after a READ or a WRITE macro-instruction is encountered in a processing program. A READ or WRITE routine partially completes a channel program using parameters from the data event control block (DECB), and passes the DECB, together with the input/output block (IOB), to an end-of-block routine. (For a description and diagram of the relationship of the DECB, the IOB, the data control block (DCB), channel program, and buffer, refer to Appendix C.)

An end-of-block routine provides device oriented data for the channel program. If normal channel-program scheduling is used, the routine passes control to the I/O supervisor (via an EXCP macro-instruction) to cause scheduling of the buffer. If chained channel-program scheduling is used, it attempts to add the present channel program to the last one in the chain of scheduled channel programs. If it is successful, control returns to the processing program. If it is unsuccessful, control passes to the I/O supervisor (via an EXCP macro-instruction). (For a detailed description of the end-of-block routines refer to "Queued Sequential Access Method Routines" section in this publication.)

A CHECK routine receives control from the processing program via a CHECK macro-instruction. A CHECK routine examines the DECB to determine the status of the channel program. If the channel program executed normally, control returns to the processing program. However, if the channel program executed with an error, control passes from the CHECK routine to the SYNAD/EOV executor (IGC0005E) for processing of error conditions. For permanent errors, control returns to the CHECK routine, and it then passes control to the processing program's SYNAD routine. (For EOV conditions, control passes to the EOV routines.)

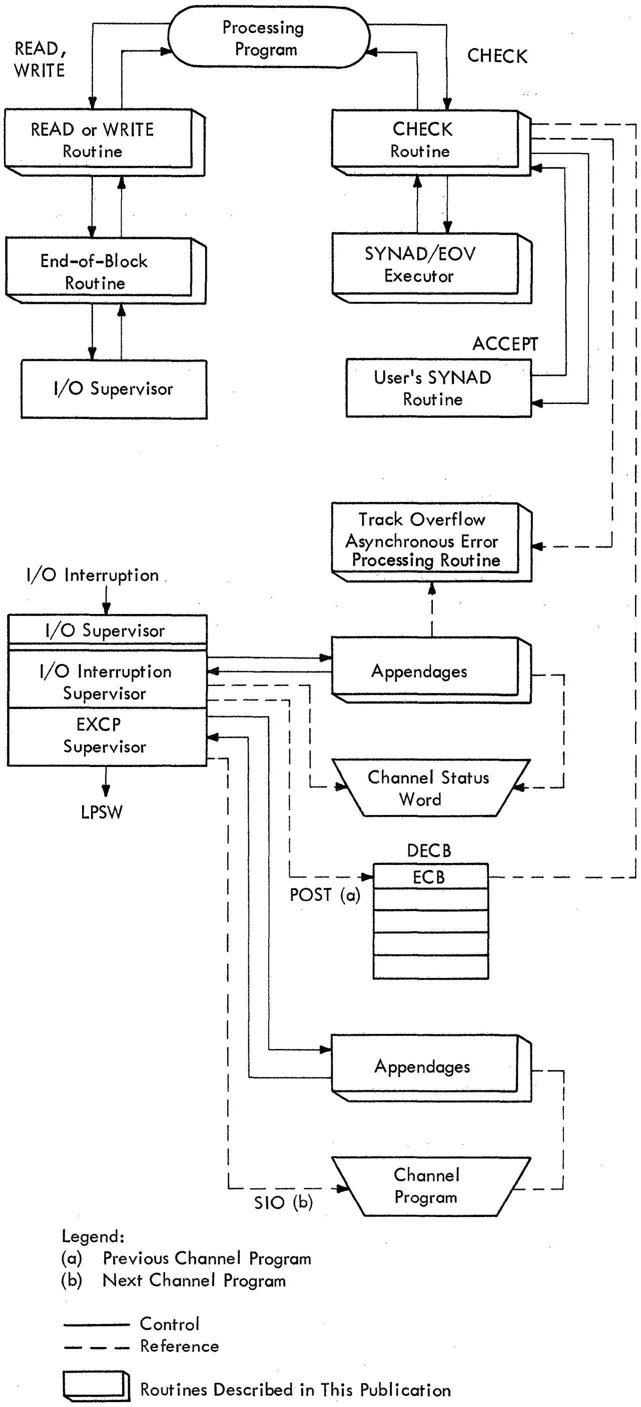


Figure 9. Flow of Control in BSAM

Table 16. Flow of Control of BSAM Routines

Routine Passing Control	Condition	Routine Receiving Control
Processing Program	READ or WRITE Macro-instruction	READ or WRITE
READ or WRITE	Branch instruction	End-of-block
End-of-block	EXCP Macro-instruction	I/O supervisor
I/O supervisor	End	End-of-block
End-of-block	End	READ or WRITE
READ or WRITE	End	Processing Program
Processing Program	CHECK Macro-instruction	CHECK
CHECK	Channel program not yet executed	WAIT
CHECK	Channel program executed with error	SYNAD/EOV Executor
CHECK	Channel program executed without error	Processing Program
Supervisor	I/O interruption	I/O supervisor
I/O supervisor	Appendage exit condition	Appendage
Appendage	End	I/O supervisor
I/O supervisor	End	Supervisor

The asynchronous error processing routine (described in the "Queued Sequential Access Methods Routines" section of this publication) gains control by being scheduled by an appendage. The routine processes permanent error conditions that are encountered by a channel program for input data with track overflow record format. The routine establishes the address of the segment beyond the one in error.

An appendage receives control from the I/O supervisor and returns control to the I/O supervisor. Some appendages operate with the I/O interruption supervisor, and others operate with the EXCP supervisor. (Appendages are described in the "Queued Sequential Access Method Routines" section of this publication.)

Control routines (not shown in Figure 9) permit the processing program to control the positioning of auxiliary storage devices. They receive control when the CNTRL (Printer, Tape, Card Reader), PRTOV, NOTE, POINT or BSP macro-instructions are encountered in a processing program. The track balance routine receives control from a WRITE routine or the track overflow end-of-block routine.

Appendix A contains decision tables that show for each type of routine, the processing characteristics that differentiate the routines within that type.

READ AND WRITE ROUTINES

A READ or WRITE routine receives control when the processing program issues a READ or a WRITE macro-instruction. The READ and WRITE routines used with data sets organized for the sequential or partitioned access methods pass control to the end-of-block routines, which in turn pass control to the I/O supervisor. The WRITE routines used to create data sets organized for later access by basic direct-access method (BDAM) routines, include the end-of-block function within themselves, and so pass control to the I/O supervisor directly. A

READ or WRITE routine processes parameters set by the processing program in the DECB, to permit scheduling of the next channel program.

There are six READ, WRITE routines. Table 17 lists the routines available and the conditions that cause a particular module to be used. The OPEN executor selects one of these routines, loads it, and puts its address into the DCBREAD/WRITE field. The table shows, for example, that module IGG019BH is selected and loaded if Update and the READ macro-instruction are specified.

Table 17. Module Selector - READ and WRITE Modules

Access Conditions	Selections							
Input or	X		X	X				
Output or		X				X	X	X
Inout, Outin	X	X						
Update					X			
READ	X		X	X	X			
WRITE		X						
WRITE (LOAD) (Create-BDAM)						X	X	X
Paper tape character conversion			X	X				
Fixed-length record format			X			X		X
Undefined-length record format or				X			X	
Variable-length record format							X	
Track Overflow								X
READ,WRITE Modules								
IGG019BA	X	X						
IGG019BF			X	X				
IGG019BH					X			
IGG019DA						X		
IGG019DB							X	
IGG019DD								X

READ/WRITE MODULE IGG019BA

Module IGG019BA completes the channel program to be scheduled next, and relates control blocks used by the I/O supervisor to the channel program. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Input, Output, Inout, or Outin

and the DCB specifies:

- READ or WRITE.

The module operates as follows:

- It receives control when a READ or WRITE macro-instruction is encountered in a processing program.
- It enters the address of the IOB into the DECB to permit the CHECK routine later to test execution of the channel program.
- It completes the channel program by inserting the buffer address from the DECB, and the length from either the DECB (for undefined-length records), the DCB (for fixed-length records, and for input of variable-length records), or the record itself (for output of variable length records).
- If a block is to be written on a direct-access storage device, the module tests the DCBOFLGS field in the DCB to establish the validity of the value in the DCBTRBAL field.
- If the DCBTRBAL value is valid, or if a block is to be written on a device other than direct-access storage, or if a block is to be read from any device, the module passes control to an end-of-block routine.
- If the DCBTRBAL value is not valid (that is, the preceding operation was a READ, POINT, or OPEN for MOD) the module issues an SVC 25 instruction to pass control to BSAM control module IGC0002E to obtain a valid track balance. When control returns to this module, it passes control to an end-of-block routine.

READ MODULE IGG019BF (PAPER TAPE CHARACTER CONVERSION)

Module IGG019BF completes a channel program to read paper tape, awaits its execution, and converts the paper tape characters into EBCDIC characters. The OPEN executor selects and loads this module (and one of the code conversion modules listed in Appendix D) if the DCB specifies:

- READ
- Fixed-length or undefined-length record format
- Paper tape.

The module operates as follows:

- It receives control when a READ macro-instruction is encountered in a processing program.
- It enters the address of the IOB into the DECB, to permit the CHECK routine to test execution of the channel program.
- It completes the channel program by inserting the buffer address from the DECB, and the length value from the field DCBBLKSI (for fixed-length record format) or the DECB (for undefined-length record format).
- It passes control to the end-of-block routine.
- When control returns from the end-of-block routine, the module issues a WAIT macro-instruction to await execution of the channel program.
- It converts each character in the buffer until one of the following conditions is met, with the stated effect:

  Conversion has provided the number of characters specified in the length value: The module returns control to the processing program.

  All the characters read have been converted, but into a smaller number of characters. (Some input character codes have no corresponding EBCDIC translation in a specific code conversion module. Therefore, after conversion of all characters in the buffer, the number of converted characters may be less than the length value): The module completes a channel program for the number of additional characters needed to fill the buffer, passes control to the end-of-block routine which issues the EXCP macro-instruction to schedule the channel program, and issues a WAIT macro-instruction for the channel program. When control returns, the module resumes converting characters.

  An end-of-record character is encountered (undefined-length record format only): The module returns control to the processing program.

  The tape is exhausted: The module returns control to the processing program.

  A paper tape reader-detected error character is encountered: If necessary because of compression, the module moves the character to the left (without conversion), and returns control to the processing program.

- If one of the characters in the buffer is an undefined character, the module converts the character to the hexadecimal character FF, sets an indication of this condition in the IOB for the paper tape CHECK routine, and continues conversion until one of the other conditions is met.

Appendix D lists the modules composed of the tables used for code conversion.

READ/WRITE MODULE IGG019BH (UPDATE)

Module IGG019BH ascertains whether a buffer supplied by the processing program is to be written from or read into, and causes a corresponding BSAM Update channel program to be executed. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Update

and the DCB specifies:

- READ.

The module operates as follows:

- It gains control when the processing program uses a READ or a WRITE macro-instruction.
- If data is to be read into a buffer, the module flags the IOB for a Read operation, sets it to point to the Read channel program, and copies the length and buffer address from the DECB or the DCB into the Read CCW.
- If data is to be written from a buffer, the module flags the IOB for a Write operation, sets it to point to the Write channel program, copies the auxiliary storage address from the DCBFDAD field into the IOBSEEK field and completes the length and buffer address entries in the Write CCW.

- The module passes control to end-of-block module IGG019CC. On return of control from that module, it returns control to the processing program.

WRITE MODULE IGG019DA (CREATE-BDAM)

Module IGG019DA writes, for a data set later to be processed by BDAM, fixed-length data blocks, fixed-length dummy blocks, and record-zero blocks. The OPEN executor selects and loads this module if the DCB specifies:

- WRITE (LOAD)
- Fixed-length record format.

The module operates as follows:

- It receives control from the processing program when it encounters a WRITE macro-instruction and also from the EOV/new volume executor after the end-of-volume routine of I/O support has obtained another extent.

- It connects the next available IOB to the DCB and the DECB.

- It determines, in the same manner as end-of-block routine IGG019CD, whether this block fits on the current track and updates the DCBTRBAL field.

- If this is neither the first nor the last block of a track, the module updates the full device address (FDAD) in the DCB and the IOB and issues an EXCP macro-instruction. It then returns control to the processing program or the EOV/new volume executor (whichever it received control from).

- If this is the last block of a track (that is, no other block will fit on the track except the present block), the module updates the full device address (FDAD) in the DCB and the IOB, expands the channel program to write the record-zero block for that track as well as the last data block, and issues an EXCP macro-instruction. The module then returns control to the routine from which it received control.

- If this is the first block of a new track and there is another track in the allocated extent, the module finds the next track in the allocated extent, updates the full device address (FDAD) in the DCB and the IOB, and issues an EXCP macro-instruction. It then returns control to the routine from which it received control.

- If this is the first block of a new track and there is no other track in the allocated extent, the module sets an EOV condition indication and returns control to the processing program.

WRITE MODULE IGG019DB (CREATE-BDAM)

Module IGG019DB writes, for a data set thereafter to be processed by BDAM, variable-length and undefined-length blocks and record-zero blocks. The OPEN executor selects and loads this module if the DCB specifies:

- WRITE (LOAD)
- Variable-length or undefined-length record format.

The module essentially consists of two routines: one to write data blocks; one to write record-zero blocks.

To write a data block for BDAM, the routine operates as follows:

- It receives control from the processing program when it encounters a WRITE-SF macro-instruction and also from EOV/new volume executor (to write the block not written into the previous volume) after the end-of-volume routine of I/O support has obtained another extent.

- It determines whether this block fits on the current track in the same manner as end-of-block routine IGG019CD and updates the DCBTRBAL field.

- If one of the following conditions exists, it returns control (without any further processing) to the processing program or to the EOV/new volume executor (whichever it received control from):

  A block other than the first block on a track is to be written, but it does not fit on the balance of the track.

  The first block is to be written on a track, but the allocated extents are exhausted. (For this condition, the module sets an EOV condition indication before it returns control.)

- If either of the following conditions exists, the module updates the full device address (FDAD) in the DCB, the IOB, and the channel program, issues an EXCP macro-instruction and then returns control to the routine from which control was received:

A block other than the first block on the track is to be written and it fits on the balance of the track.

The first block is to be written on a track and there is another track in the allocated extents.

- It returns control to the processing program or the end-of-volume routine.

To write a record-zero block for BDAM, the routine operates as follows:

- It receives control when a WRITE-SZ macro-instruction is encountered in the processing program, or after the end-of-volume routine has obtained another extent.
- It updates the record-zero area and the channel program to write the record-zero block and issues an EXCP macro-instruction. The routine returns control to the processing program or to the end-of-volume routine.
- If there are no data blocks on the track, the module modifies the channel program to clear the track after writing the record-zero block.

WRITE MODULE IGG019DD (CREATE-BDAM - TRACK OVERFLOW)

Module IGG019DD creates data sets (with track overflow) of fixed-length data and fixed-length dummy blocks that are subsequently to be processed by BDAM. The module segments the block, enters the segment lengths and buffer segment addresses in the channel program, updates storage addresses for the channel program, and count fields for the block to be written and for records zero of the tracks. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Output

and the DCB specifies:

- WRITE (LOAD)
- Fixed-length record format
- Track overflow.

The module operates as follows:

- It receives control from the processing program when the program finds a WRITE macro-instruction, or from the end-of-volume routine of I/O support after that routine has obtained a new volume to write out any pending channel programs. (The end-of-volume routine receives control from the CHECK routine when that routine finds that a channel program did not execute because of an end-of-volume condition.)
- If no IOB is available, it returns control to the processing program.
- If an IOB is available, it stores its address in the DCB and the DECB.
- If the block last written was the last one for this extent, the module erases the balance of the extent.
- If the block last written filled the last track used, the module obtains the address of the next track.
- It sets the IOB and its channel program to write the block onto the next available track.
- If the block does not fill the track, the module completes the count field for this record and issues an EXCP macro-instruction.
- If the block fills the track, the module sets the track-full indicator, completes record zero for this track, links the channel program that writes record zero to the channel program that writes the data record, and issues an EXCP macro-instruction.
- If the block overflows the track, the module completes record zero for this track and completes a channel program to write record zero, completes the count field and channel program for the segment that fits on the track, and constructs the identification for record one of the next track.
- It repeats the preceding until a segment is left that does not overflow a track. For the final segment, the module operates as for a block that fits on the track.
- On return of control from the I/O supervisor, the module returns control to the routine from which it was received.

CHECK ROUTINES

A CHECK routine synchronizes the execution of channel programs with that of the processing program. When the processing program issues a READ or WRITE macro-instruction, control returns to the processing program (from the READ or WRITE routine) when the channel program has been scheduled for execution or, if reading paper tape, when the buffer has been filled and the data converted. To determine the state of execution of the channel program, the processing program issues a CHECK macro-instruction; control returns to the processing program (from the CHECK routine) if the channel program was executed successfully, or if it was executed successfully after the CHECK routine caused volume-switching. For permanent errors, control passes to the processing program's SYNAD routine. Reading or writing under BSAM, the SYNAD routine may continue processing the data set by returning control to the CHECK routine; writing in the Create-BDAM mode, processing cannot be resumed.

There are four CHECK routines. Table 18 lists the routines available and the conditions that cause a particular module to be used. The OPEN executor selects one of the four routines, loads it, and places its address into the DCBCHECK field. For example, the table shows that module IGG019BG is selected and loaded if READ and paper tape character conversion is specified.

Table 18. Module Selector - CHECK Modules

Access Conditions	Selections				
Input or	X		X		
Output or		X			
Inout, Outin	X	X			
Update				X	
READ	X		X		
WRITE		X			
WRITE(LOAD) (Create-BDAM)					X
Paper tape character conversion			X		
CHECK Modules					
IGG019BB	X	X			
IGG019BG			X		
IGG019BI				X	
IGG019DC					X

CHECK MODULE IGG019BB

Module IGG019BB synchronizes the execution of the channel program to that of the processing program, and responds to any exceptional condition remaining after the I/O supervisor has posted execution of the channel program in the IOB. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Input, Output, Inout, or Outin

and the DCB specifies:

- READ or WRITE.

The module operates as follows:

- It receives control when a CHECK macro-instruction is encountered in a processing program.
- It tests the DECB for successful execution of the channel program.
- If the channel program was executed normally, the module returns control to the processing program.
- If the channel program is not yet executed, the module issues a WAIT macro-instruction.
- If the channel program encountered an error condition in its execution, the module issues an SVC 55 instruction to pass control to the SYNAD/EOV executor (IGC0005E). Two types of returns from the executor are possible:

  If the executor determines the error condition to be an EOV condition, the executor passes control to the end-of-volume routine of I/O support for volume switching. That routine passes control to the EOV/New Volume executor which reschedules the purged channel programs. That executor returns control to the CHECK module.

  If the executor determines the error condition to be a permanent error, the executor returns control to the CHECK module immediately. Control is then passed to the processing program's SYNAD routine. If the SYNAD routine returns control to CHECK routine, the routine issues a second SVC 55 instruction to pass control to the SYNAD/EOV executor (IGC0005E) again. The executor treats this as an ACCEPT error option, implements it, and returns control to the routine, which then returns control to the processing program.

CHECK MODULE IGG019BG (PAPER TAPE CHARACTER CONVERSION)

Module IGG019BG processes error conditions detected by READ module IGG019BF.

This module is loaded if the DCB specifies the READ macro-instruction and paper tape character conversion.

The module operates as follows:

- It receives control when a CHECK macro-instruction is encountered in a processing program.
- If the READ routine filled the buffer with valid characters, the CHECK module returns control to the processing program.
- If the READ routine stopped converting because of a reader-detected error character, or if the READ routine encountered an undefined character, the CHECK module passes control to the processing program's SYNAD routine.
- If control returns from the SYNAD routine, the CHECK module returns control to the processing program.
- If the channel program encountered an EOV condition, the CHECK module issues an SVC 55 instruction. Control passes to the SYNAD/EOV executor (IGC0005E), then to the end-of-volume routine of I/O support, and finally to the processing program's EODAD routine.

CHECK MODULE IGG019BI (UPDATE)

Module IGG019BI synchronizes the execution of a BSAM Update channel program to the progress of the processing program. (A BSAM Update channel program either writes data from a buffer or reads data into a buffer.) The module also causes processing of permanent errors and end-of-volume conditions. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

- Update

and the DCB specifies:

- READ.

The module operates as follows:

- It receives control when the processing program uses the CHECK macro-instruction.
- It tests the ECB in the DECB for successful execution of the channel program associated with that DECB.
- If the channel program is not yet executed, the module uses a WAIT macro-instruction.
- If the channel program has been executed normally, the module returns control to the processing program.
- If the channel program encountered an error condition in its execution the module tests to determine if the error is an EOV condition.
- If the error is an EOV condition, the module sets an indicator to show that this entry is from the CHECK module and passes control to the processing program's EODAD routine.
- If the error is not an EOV condition the module issues an SVC 55 instruction to pass control to the SYNAD/EOV executor (module IGC0005E).
- On return of control from the SYNAD/EOV executor the CHECK module passes control to the processing program's SYNAD routine. If the SYNAD routine returns control to CHECK routine, the routine issues a second SVC 55 instruction to pass control to the SYNAD/EOV executor (IGC0005E) again. The executor treats this as an ACCEPT error option, implements it, and returns control to this routine, which then returns control to the processing program.

CHECK MODULE IGG019DC (CREATE-BDAM)

Module IGG019DC synchronizes the execution of the channel program (to write a block for a BDAM data set) to the progress of the processing program, and responds to exceptional conditions encountered in the execution of the channel program. The OPEN executor selects and loads this module if the DCB specifies:

- WRITE (LOAD).

The module operates as follows:

- It receives control when the processing program uses the CHECK macro-instruction.

- If the channel program is not yet executed, the module issues a WAIT macro-instruction.

- If the channel program executed without error, the module returns control to the processing program.

- If the execution of the channel program encountered a permanent error condition, the module passes control to the processing program's SYNAD routine. If control is returned from the SYNAD routine, or if there is no SYNAD routine, the module issues an ABEND macro-instruction.

- If the WRITE routine encountered an EOV condition (and, therefore did not request scheduling of the channel program for execution), this module passes control to the SYNAD/EOV executor (IGC0005E) by issuing an SVC 55 instruction. On return of control this module tests for completion of the channel program.

BSAM CONTROL ROUTINES

A control routine receives control when a control macro-instruction (for example, CNTRL, NOTE, POINT, BSP) is used in a processing program or in another control routine. BSAM control routines (which include those available in QSAM) pass control to the I/O supervisor, another control routine, or return control to the processing program directly. BSAM control routines cause the physical or logical positioning of auxiliary storage devices.

There are three types of BSAM control routines:

- Routines that are loaded into processing program main storage by the OPEN executor (CNTRL, NOTE/POINT).

- Routines that are loaded into supervisory transient area main storage by an SVC instruction in a processing program macro-expansion or in another control routine (BSP, Track Balance).

- Routines that are in-line macro-expansions in the processing program (PRTOV).

Routines that are loaded by the OPEN executor are mutually exclusive; that is, only one of them can be used with one DCB. The PRTOV macro-expansions result in instructions that set or test bits that cause branching in either the processing program or in an appendage.

Tables 19, 20, and 21 list the various kinds of control routines and the parameters that cause them to gain control. Table 19 shows the access condition options that cause the OPEN executor to load a control routine for use with a DCB. Table 20 lists the SVC instructions that cause a control routine to be loaded at execution time. Table 21 lists the different macro-expansions constructed by the assembler.

Table 19. Module Selector - Control Modules Selected and Loaded by the OPEN Executor

Access Conditions	Selection						
NOTE/POINT	X	X		X	X		
Update, Track Overflow, or				X			
Chained Scheduling				X	X		
CNTRL			X			X	X
Direct-Access Storage	X			X			
Magnetic Tape		X	X		X		
Printer						X	
Card Reader							X
Control Modules							
IGG019BC	X						
IGG019BD		X					
IGG019BE			X				
IGG019BK				X			
IGG019BL					X		
IGG019CA[1]						X	
IGG019CB[1]							X

[1] These routines are also used in QSAM; see that section for description of these routines.

Table 20. Control Modules Loaded at Execution Time

SVC No.	Macro-Instruction	Function	Module No.
25	(none)	Establish valid track balance Erase balance of extent for track overflow	IGC0002E
69	BSP	Device Independent Backspace (tape, direct-access)	IGC0006I

Table 21. Control Routines That Are Macro-Expansions[1] [2]

Macro-Instruction	Number of Macro-Expansions
PRTOV - User exit	1
PRTOV - No user exit	1

[1] These routines are also used in QSAM; see that section for a description of the routines.

[2] This table duplicates Table 15; it is repeated here to identify all control routines available in BSAM.

CONTROL MODULE IGG019BC (NOTE, POINT - DIRECT-ACCESS)

The OPEN executor selects and loads this module if the DCB specifies:

- POINT
- Direct-access storage device.

The module consists of two routines: NOTE and POINT.

NOTE Routine

The NOTE routine in module IGG019BC converts the full direct-access device address (FDAD) for the last block read or written, to a relative address (of the form TTR), and presents that value to the processing program.

The NOTE routine operates as follows:

- It receives control when a NOTE macro-instruction is encountered in a processing program.
- It obtains the FDAD value used by the channel program last executed. The address is found in either the IOB or the DCB depending upon which macro-instruction the last channel program implemented.
- If the macro-instruction was READ and more than one buffer is used, the channel program last executed placed the FDAD value into the IOBSEEK field in the IOB.
- If the macro-instruction was READ and a single buffer is used, the channel program last executed placed the FDAD value into the DCBFDAD field of the DCB.
- If the macro-instruction was WRITE, the end-of-block routine placed the FDAD value into the DCBFDAD field.
- It issues a BALR instruction to pass control to the IECPRLTV routine, which converts full addresses into relative addresses.
- It returns the address and control to the processing program.

POINT Routine

The POINT routine in module IGG019BC converts a relative address (of the form TTRZ) to the full direct-access device address (FDAD) used by the next channel program to read or write the block noted.

The POINT routine operates as follows:

- It receives control when a POINT macro-instruction is encountered in a processing program.
- It issues a BALR instruction to pass control to the IECPCNVT routine. That routine converts the relative address to the full address and returns control to the POINT routine. If the processing program passed an invalid relative address, the routine sets the DCBIFLGS and IOBECBCC fields to show that an addressing error occurred, before returning control. (The CHECK routine finds the error and processes accordingly.)

- It establishes the actual value to be used by the next channel program by testing the fourth byte of the relative address (TTRZ). If the value of Z is zero, the full address is decremented by one; if Z is one, the address calculated by the IECPCNVT routine is left unchanged. (For an explanation of how the value of Z is set, refer to the description of the POINT macro-instruction in the publication IBM System/360 Operating System: Control Program Services.)
- It inserts the value in the DCBFDAD and IOBSEEK fields, sets the DCBOFLGS field to show that the contents of the DCBTRBAL field are no longer valid, and returns control to the processing program.

CONTROL MODULE IGG019BD (NOTE, POINT - MAGNETIC TAPE)

The OPEN executor selects and loads this module if the DCB specifies:

- POINT
- Magnetic Tape.

This module consists of two routines: NOTE and POINT.

NOTE Routine

The NOTE routine in module IGG019BD presents the contents of the DCBBLKCT field of the DCB to the processing program and returns control to the processing program.

POINT Routine

The POINT routine in module IGG019BD positions the tape at the block for which NOTE was issued.

The POINT routine operates as follows:

- It receives control when a POINT macro-instruction is encountered in a processing program.
- It constructs a channel program to read forward or backward one block.
- It passes the channel program for execution the number of times required to position the tape at the desired block.
- It follows the last Read channel program by a NOP channel program to obtain device end information for the last spacing operation.
- It returns control to the processing program, unless a tape mark, load point, or permanent error is encountered in one of the executions of the Read channel program. In that case, the routine sets the DCBIFLGS field to indicate a permanent error, before returning control to the processing program. (Subsequent processing by the READ or WRITE routine to cause scheduling of channel programs for execution will result in their not being scheduled. On the next entry into the CHECK routine, it detects and processes the error condition.)

CONTROL MODULE IGG019BE (CNTRL: SPACE TO TAPE MARK, SPACE TAPE RECORDS)

Module IGG019BE positions magnetic tape at a point within the data set specified by the CNTRL macro-instruction. The OPEN executor selects and loads this module if the DCB specifies:

- CNTRL
- Magnetic Tape.

The module consists essentially of two routines: One for spacing forward or backward to the tape mark (the FSM/BSM routine), and one for spacing forward or backward a number of tape records (the FSR/BSR routine).

The FSM/BSM routine operates as follows:

- It receives control when a CNTRL macro-instruction is encountered in a processing program.
- It constructs a channel program to space to the tape mark in the desired direction.
- It issues an EXCP macro-instruction for the FSM or BSM channel program. Control returns to the routine at channel end for the FSM/BSM channel program.
- It issues an EXCP macro-instruction for a NOP channel program to obtain device end information from the FSM/BSM channel program.
- It issues an EXCP macro-instruction for a BSR or FSR channel program to position the tape within the data set, after the FSM/BSM channel program encounters a tape-mark.

- It issues an EXCP macro-instruction for a NOP channel program again, to obtain device end information from the BSR/FSR channel program. The routine then returns control to the processing program.

The FSR/BSR routine operates as follows:

- It receives control when a CNTRL macro-instruction is encountered in a processing program.
- It constructs a channel program to space one record in the desired direction.
- It reduces the count passed by the control macro-instruction and issues an EXCP macro-instruction for the FSR or BSR channel program.
- When the count is zero, it issues an EXCP macro-instruction for a NOP channel program to obtain the device end information from the last FSR/BSR channel program. The routine then returns control to the processing program.
- If a load point is encountered during spacing, the routine returns control to the processing program.
- If a tape mark is encountered during spacing, the routine repositions the tape to a point within the data set by reverse spacing one block and returns control to the processing program.
- If a permanent error is encountered during spacing, the routine issues a BALR instruction to pass control to the SYNAD routine, if one is present; if not, it issues an ABEND macro-instruction.

CONTROL MODULE IGG019BK (NOTE, POINT - DIRECT-ACCESS - SPECIAL)

This module contains the NOTE and POINT routines for the special access conditions of chained scheduling, track overflow, and Update. The OPEN executor selects and loads this module if the DCB specifies:

- POINT
- Direct-access storage
- Chained scheduling, track overflow, or the OPEN parameter is Update.

NOTE Routine

The NOTE routine in module IGG019BK finds the full direct-access device address (FDAD) for the last block read or written, converts it to a relative address (of the form TTR), and presents that value to the processing program.

The NOTE routine operates as follows:

- It receives control when a NOTE macro-instruction is encountered in a processing program.
- It obtains the FDAD value used by the channel program last executed. The location of this address depends on which macro-instruction the last channel program implemented.
- If the macro-instruction was READ and more than one buffer is used, the channel program last executed placed the FDAD value into the IOBSEEK field in the IOB if track overflow or Update is being used, and into the ICBSEEK field if chained scheduling is used.
- If the macro-instruction was READ and only a single buffer is used the channel program last executed placed the FDAD value into the DCBFDAD field of the DCB.
- If the macro-instruction was WRITE, the end-of-block routine placed the FDAD value into the DCBFDAD field.
- It issues a BALR instruction to pass control to the IECPRLTV routine, which converts full addresses into relative addresses.
- It returns the address and control to the processing program.

POINT Routine

The POINT routine in module IGG019BK establishes the full direct-access device address (FDAD) used by the channel program to read or write the block noted.

The POINT routine operates as follows:

- It receives control when a POINT macro-instruction is encountered in a processing program.
- It issues a BALR instruction to pass control to the IECPCNVT routine. That routine converts the relative address to the full address and returns control to the POINT routine. If the processing program passed an invalid relative address, the executor sets the DCBIFLGS and the IOBECBCC fields to show that an addressing error occurred, before returning control. (The CHECK routine finds the error and processes accordingly.)

- It establishes the actual value to be used by the next channel program by testing the fourth byte of the relative address (TTRZ). If the value of Z is zero, the full address is decremented by one; if Z is one, the address calculated by the convert routine is left unchanged. (For an explanation of how the value of Z is set, refer to the description of the POINT macro-instruction in the publication IBM System/360 Operating System/360 Operating System: Control Program Services.)

- It inserts the value into the DCBFDAD and IOBSEEK fields if track overflow or Update is being used, and also into the ICBSEEK field if chained scheduling is used. It sets the DCBOFLGS field to show that the contents of the DCBTRBAL field are no longer valid, and returns control to the processing program.

CONTROL MODULE IGG019BL (NOTE, POINT - MAGNETIC TAPE - CHAINED SCHEDULING)

Module IGG019BL is selected and loaded by the OPEN executor if the DCB specifies:

- POINT
- Magnetic Tape
- Chained scheduling.

The module consists of two routines: NOTE and POINT.

NOTE Routine

The NOTE routine in module IGG019BL presents the contents of the DCBBLKCT field of the DCB to the processing program and returns control to the processing program.

POINT Routine

The POINT routine in module IGG019BL positions the tape at the block for which NOTE was issued. It operates as follows:

- It receives control when a POINT macro-instruction is encountered in a processing program.

- A channel program is constructed to read forward or backward one block.

- The channel program is passed for execution the number of times required to position the tape at the desired block.

- The last spacing channel program is followed by a NOP channel program to obtain device end information for the last spacing operation.

- Control is returned to the processing program, unless a tape mark, load point, or permanent error is encountered in the execution of one of the channel programs. In that case, the routine sets the DCBIFLGS field to indicate a permanent error before returning control to the processing program. (Subsequent attempts by the READ or WRITE routine to cause scheduling of channel programs for execution will result in their not being scheduled. On the next entry into the CHECK routine, that routine detects and processes the condition.)

CONTROL MODULE IGC0002E (SVC 25 - TRACK BALANCE, TRACK OVERFLOW ERASE)

Module IGC0002E consists of two routines that erase either a part of one track or several tracks. The track balance routine determines the available space by erasing the remainder of the track; the track overflow erase routine erases tracks at the end of each extent on which there are no data fields for blocks of the data set to which the extent belongs. The routine is used when a block in a data set with track overflow record format would span extents.

This module is loaded at execution time into supervisor transient area main storage if either READ/WRITE module IGG019BA or end-of-block module IGG019C2 arrives at an SVC 25 instruction.

Track Balance Routine

The track balance routine establishes a valid value for the DCBTRBAL field of a DCB opened for output to a direct-access device, when the field value has been invalidated by a preceding READ, POINT, or OPEN for MOD macro-instruction.

The routine operates as follows:

- It receives control after it is loaded.

- It constructs, and issues an EXCP macro-instruction for, a channel program with the Erase command and a count exceeding the track capacity. The erase operation begins following the block just read or on the block pointed at.

- It determines the actual track balance by subtracting the residual count in the channel status word (CSW) from the count used in the channel program, and inserts the difference in the DCBTRBAL field of the DCB.

- It returns control to the WRITE routine.

Track Overflow Erase Routine

The track overflow erase routine erases the space on a direct-access storage device that lies between the last block to be written into the current extent and the end of that extent. If track overflow end-of-block routine IGG019C2 finds that the next segment of a block falls on a track beyond the present extent, that end-of-block routine uses the SVC 25 instruction to pass control, and the channel program, to this routine.

The routine operates as follows:

- It receives control when it is loaded.

- It substitutes Erase commands for the Write commands in the channel program associated with the present IOB.

- It issues an EXCP macro-instruction to cause execution of the channel program and a WAIT macro-instruction for its completion.

- It returns control to the track overflow end-of-block routine, irrespective of any errors in the execution of the channel program.

CONTROL MODULE IGC0006I (SVC 69 - BSP)

Module IGC0006I backspaces the data set one block, whether the data set is on a magnetic tape or direct-access device.

The expansion of the macro-instruction BSP includes an SVC 69 instruction which causes the module to be loaded and entered. The module essentially consists of two parts, one for magnetic tape and one for direct-access devices.

For magnetic tape, the module operates as follows:

- It receives control after it is loaded.

- It constructs and issues an EXCP macro-instruction for a channel program to backspace one block.

- It constructs and issues an EXCP macro-instruction for a NOP channel program to obtain device end information from the backspace channel program.

- If the backspace channel program executed normally, the module sets register 15 to zero and returns control to the processing program.

- If the channel program executed with an error other than unit exception, the module sets the DCBIFLGS field to indicate a permanent error. (The CHECK macro-instruction, following the next READ or WRITE macro-instruction, causes the CHECK routine to pass control to the processing program's SYNAD routine.)

- If the backspace channel program executed with a unit exception, the module constructs and issues an EXCP macro-instruction for a channel program to forward space the tape one block. It next constructs and issues a NOP channel program to obtain device end information from the forward space channel program. When channel end for the NOP channel program occurs, the module returns control to the processing program with register 15 set to an error code.

For direct-access devices, the module operates as follows:

- It receives control after it is loaded.

- It decrements the DCBFDAD field in the DCB to the preceding block address, across tracks, cylinders, or extents.

- It sets the DCBOFLGS field to show that the DCBTRBAL field value is invalid.

- If a valid preceding DCBFDAD value has been established, the module returns control to the processing program with register 15 set to zero.

- If there is no valid preceding DCBFDAD value (because the processing program attempts to backspace beyond the first block), the module returns control to the processing program with register 15 set to an error code.

- If a permanent error is encountered when reading the count fields (to establish the preceding DCBFDAD field value), the DCBIFLGS field value is set to indicate a permanent error. (The CHECK routine, following the next READ or WRITE macro-instruction, causes control to pass to the processing program's SYNAD routine.)

A partitioned data set has a directory and members. The directory is read and written using BPAM routines; the members are read and written using BSAM routines. (Refer to the BSAM portion of this publication.) A processing program using BPAM routines for input from the directory is presented with the address of a member in a channel program or in a table; fcr a processing program using BPAM for output to a directory, the routines determine the address of the member and record that address in the directory.

BPAM ROUTINES

BPAM routines store and retrieve entries in the directory and convert between relative and absolute auxiliary storage addresses. Directory entries are entered and found by constructing channel programs that search the directory for appropriate entry blocks and by locating an equal, or higher, entry within the block. Address converting routines refer to the data extent block (DEB) to determine the address value complementary to the given value.

BPAM routines (see Table 22) differ from BSAM and QSAM routines in that BPAM routines are not loaded at OPEN time; the STOW routine is loaded at execution time, all the coding for FIND (C option) is a macro-expansion, and the FIND (D option)/BLDL routine and the converting routines are in resident main storage. Table 22 shows how these routines gain control.

STOW MODULE IGC0002A (SVC 21)

Module IGC0002A finds entries in BPAM directory entry blocks and keeps the directory left-justified after entries have been inserted or deleted.

The expansion of the STOW macro-instruction includes an SVC 21 instruction that causes this module to be loaded and to gain control. The STOW macro-instruction is issued in one of two ways:

- Explicitly by a processing program using BPAM for output.
- Implicitly by a processing program using BSAM, QSAM, or BPAM for output, when issuing a CLOSE macro-instruction to a DCB opened for a member of a partitioned data set.

Table 22. BPAM Routines Residence

BPAM Routines	Module Number	Residence	Instruction Passing Control
STOW	IGC0002A	Supervisory Transient Area	SVC 21
FIND (C Option)	(Macro Expansion)	Processing Program Area	FIND (C Option)
FIND (D Option)	IECPFIND,IECPFND1	Supervisory Resident Area	SVC 18
BLDL	IECPFIND,IECPFND1	Supervisory Resident Area	SVC 18 or BAL IECPBLDL
Convert TTR	IECPFIND,IECPFND1	Supervisory Resident Area	BAL IECPCNVT
Convert MBBCCHHR	IECPFIND,IECPFND1	Supervisory Resident Area	BAL IECPRLTV

The module operates as follows:

- It receives control when it is loaded.

- If an ADD (Not ALIAS) or a REPLACE (Not ALIAS) option is specified, the module writes an end-of-data set mark (zero-length data block) at the end of the member. The module then stores, for use at the next entry into the STOW module, the relative address of the next block to be written, in the DCBRELAD field of the DCB. (The OPEN routine determines the first relative address for the first entry to this module.)

- For any option, the module searches the directory for an entry block with a key equal to or higher than the member name, and reads that entry block into the input buffer.

- The module compares the entries in the entry block to the member name in the instruction operand. Entries whose value is lower than that of the member name are moved to the output buffer.

- For entries that equal the member name, the module checks to determine whether the REPLACE, the CHANGE, or the DELETE option is specified.

- If the REPLACE option is specified, the module moves the new entry from the work area to the output buffer, skips the present entry, and moves the remaining entries to the output buffer. It issues an EXCP macro-instruction to write the updated entry block into the directory.

- If the CHANGE option is specified, the module moves the present entry less the present name to the new entry work area. To enter the new entry in its proper entry block, the routine continues as though the ADD option were specified.

- If the DELETE option is specified, the module skips the present entry and moves the remaining entries to the output buffer. The module now shifts the balance of the entries in the directory to the left by constructing the necessary channel programs. It reads a block, shifts entries into the remaining space of the preceding block, writes the completed entry block, and starts the next block.

- For entries that are higher than the member name, the module checks to determine whether the ADD option is specified.

- If the ADD option is specified, the module moves the new entry from the work area to the output buffer before moving the high entry and those following it. The module then shifts to the right all entries following the added entry by constructing the channel programs necessary alternately to write and read entry blocks. The module writes the full block, moves the remaining entries to the output buffer, reads another entry block, and then completes and writes the output buffer.

- On completion of all channel programs necessary for the specified option, the routine returns control to either the processing program, or the CLOSE routine.

FIND (C OPTION) MACRO-EXPANSION

This coding causes translation of the relative address into a full device address (FDAD) and its insertion into the next IOB.

The macro-expansion produces object code that places the relative address in the DCBRELAD field in the DCB and issues a BALR instruction to pass control to the POINT routine.

RESIDENT MODULE IECPFIND

Unless BLDLTAB is specified for the RESIDNT option of the SUPRVSOR macro-instruction in the system generation (SYSGEN) program, this module is link-edited at SYSGEN time with other modules to make up the resident nucleus. (If BLDLTAB is specified, module IECPFND1 is used.)

The routines composing the module gain control through an SVC 18 instruction in a processing program or a BALR instruction in a control program. A FIND (D Option) or BLDL macro-instruction expansion generates an SVC 18 instruction which causes control to pass to CSECT IGC018, the entry point for the FIND (D Option) and BLDL routines. Control programs may use a BALR instruction and the address found in the communications vector table (CVT) for entry points IECPBLDL, IECPCNVT, and IECPRLTV to pass control to the respective routines.

FIND (D Option) Routine - Entry Point and CSECT Name: IGC018 (SVC 18)

The FIND (D Option) routine finds the relative address of the member named in the macro-instruction. It then causes the relative address to be converted into the full device address (FDAD) and to be loaded into the DCBFDAD and IOBSEEK fields. The routine operates as follows:

- It searches the directory for an entry block with a key equal to, or higher than, the given member name.
- It reads that entry block into main storage and searches the entry block for the matching entry.
- It enters the relative address stated in the entry into the DCBRELAD field in the DCB and issues a BAL instruction to pass control to the POINT routine. Control returns to the processing program.

BLDL Routine - Entry Points: IECPBLDL, IGC018 (SVC 18)

The BLDL routine completes a BLDL table with the directory entry for each of the members named in the BLDL table. The routine operates as follows:

- It searches the directory for an entry block with a key equal to, or higher than, the given member name.
- It reads that block into main storage and searches the entry block for the matching entry.
- It moves the entry into the processing program's BLDL table, obtains the next name to be matched, and returns to the beginning of the routine.
- When the BLDL table has been completed, the routine returns control to the processing program.

Convert Relative-to-Full Address Routine - Entry Point: IECPCNVT

Converting routine IECPCNVT accepts, in register 0, a relative address (of the form TTR) for direct-access devices and presents the corresponding full device address (of the form MBBCCHHR) at the location shown by register 2.

The routine operates as follows:

- For each extent, the module reduces the amount TT by the number of tracks in the extent. When the balance is negative, the proper extent has been reached.
- It determines the full device address for the specified relative value.

Convert Full-to-Relative Address Routine - Entry Point: IECPRLTV

Converting routine IECPRLTV accepts, from the location shown by register 2, a full device address (of the form MBBCCHHR) for direct-access devices and presents the corresponding relative address (of the form TTR) in register 0.

The module totals the number of tracks per extent for the (M - 1) extents. For extent M, it adds the number of tracks entered into the extent.

RESIDENT MODULE IECPFND1

If BLDLTAB is specified for the RESIDNT parameter of the SUPRVSOR macro-instruction when the system is generated, this module is link-edited at SYSGEN time with other modules to make up the resident nucleus. (If BLDLTAB is not specified, module IECPFIND is used.) At initial program loading (IPL) time, the nucleus initialization program (NIP) constructs a resident BLDL table from SYS1.LINKLIB directory entries. That table is the one referred to by the FIND and BLDL routines in this module.

The routines composing the module gain control through an SVC 18 instruction in a processing program or a BALR instruction in a control program. A FIND (D Option) or BLDL macro-instruction expansion generates an SVC 18 instruction which causes control to pass to CSECT IGC018, the entry point for the FIND (D Option) and BLDL routines. Control programs may use a BALR instruction and the address found in the communications vector table (CVT) for entry points IECPBLDL, IECPCNVT, and IECPRLTV to pass control to the respective routines.

FIND (D Option) Routine - Entry Point and CSECT Name: IGC018 (SVC 18)

The FIND (D Option) routine finds the relative address of the member named in the macro-instruction. It then causes the relative address to be converted into the full device address (FDAD) and to be loaded into the DCBFDAD and IOBSEEK fields. The routine operates as follows:

- If SYS1.LINKLIB is the referenced library, it scans the resident BLDL table for an entry that matches the given member name.
- If SYS1.LINKLIB is not the referenced library, or if the name is not in the table, it searches the directory for an entry block with a key equal to, or higher than, the given member name. It reads that entry block into main storage and searches the entry block for the matching entry.
- If the name is in the table, or after finding the matching entry in an entry block read in, it enters the relative address stated in the entry into the DCBRELAD field in the DCB.
- It issues a BAL instruction to pass control to the POINT routine.
- It returns control to the processing program.

BLDL Routine - Entry Points: IECPBLDL, IGC018 (SVC 18)

The BLDL routine completes a BLDL table with the directory entry for each of the members named in the BLDL table. The routine operates as follows:

- If SYS1.LINKLIB is the referenced library, it scans the resident BLDL table for an entry that matches the given member name.
- If SYS1.LINKLIB is not the referenced library, or if the name is not in the table, it searches the directory for an entry block with a key equal to, or higher than, the given member name. It reads that block into main storage and searches the entry block for the matching entry.
- If the name is in the table, or after finding the matching entry in an entry block read in, it moves the entry into the processing program's BLDL table, obtains the next name to be matched, and returns to the beginning of the routine.
- When the BLDL table has been completed, the routine returns control to the processing program.

Convert Relative-to-Full Address Routine - Entry Point: IECPCNVT

Converting routine IECPCNVT accepts, in register 0, relative addresses (of the form TTR) for direct-access devices and presents the corresponding full device addresses (of the form MBBCCHHR) at the location shown by register 2.

The routine operates as follows:

- For each extent, the routine reduces the amount TT by the number of tracks in the extent. When the balance is negative, the proper extent has been reached.
- It determines the full device address for the specified relative value.

Convert Full-to-Relative Address Routine - Entry Point: IECPRLTV

Converting routine IECPRLTV accepts, from the location shown by register 2, a full device address (of the form MBBCCHHR) for direct-access devices and presents the corresponding relative address (of the form TTR) in register 0.

The routine totals the number of tracks per extent for the (M - 1) extents. For extent M, it adds the number of tracks entered into the extent.

Sequential access method executors are routines that receive control from, pass control to, or return control to I/O support routines. (For a description of I/O support routines refer to the publication IBM System/360 Operating System: Input/Output Support, Program Logic Manual, Form Y28-6609.) Table 23 shows the sequence of control between executors and other routines. Executors perform processing unique to an access method when a data control block is being opened or closed, or an end-of-volume condition is being processed. These executors (used for QSAM, BSAM, and BPAM) are of five types:

- OPEN executor
- CLOSE executor
- SYNAD/EOV executor
- EOV/new volume executor
- FEOV executor.

Executors differ from other access method routines in that they are executed from the supervisory transient area. It is the OPEN executor that loads the access method routines into the processing program area for later use during processing program execution.

The OPEN executor is entered from the OPEN routine of I/O support, and returns control to that routine. (See Figure 10.) It constructs the data extent block (DEB), the input/output blocks (IOB), the channel programs, and, if chained channel-program scheduling is used, interruption control blocks (ICB). It selects and load the access method routines to be used with the data control block (DCB) being opened.

The CLOSE executor is entered from the CLOSE routine of I/O support, and returns control to it. The executor handles any pending channel programs and releases the main storage used by the IOBs (and ICBs) and channel programs.

The SYNAD/EOV executor is entered when synchronizing or CHECK routine finds that a permanent I/O error or end-of-volume (EOV) condition was encountered during the execution of a channel program. The executor passes control to the end-of-volume routine of I/O support, or executes the error options specified by the processing program. The executor provides a work area in main storage for the end-of-volume routine.

The FEOV (force-end-of volume) executor is entered when an FEOV macro-instruction is encountered in a processing program. The executor handles any pending channel programs, provides a work area in main storage for the end-of-volume routine, and passes control to the end-of-volume routine of I/O support.

The EOV/new, volume executor receives control from the end-of-volume routine of I/O support. The executor causes the I/O supervisor to reschedule any channel programs not executed because of the EOV conditions.

Table 23. Sequential Access Method Executors - Control Sequence

Executor	Number	Receives Control From	Via	Passes Control To
OPEN	See Tables 24, 25,26	See Figure 10	XCTL (WTG Table)	See Figure 10
CLOSE	IGG0201A IGG0201B	CLOSE Routine	XCTL (WTG Table)	CLOSE Routine
SYNAD/EOV	IGC0005E	Synchronizing, CHECK Routines	SVC 55	EOV Routine
FEOV	IGC0003A	Processing Program	FEOV Macro-Instruction (SVC 31)	EOV Routine
EOV/new volume	IGG0551A	EOV Routine	XCTL	See Executor Description

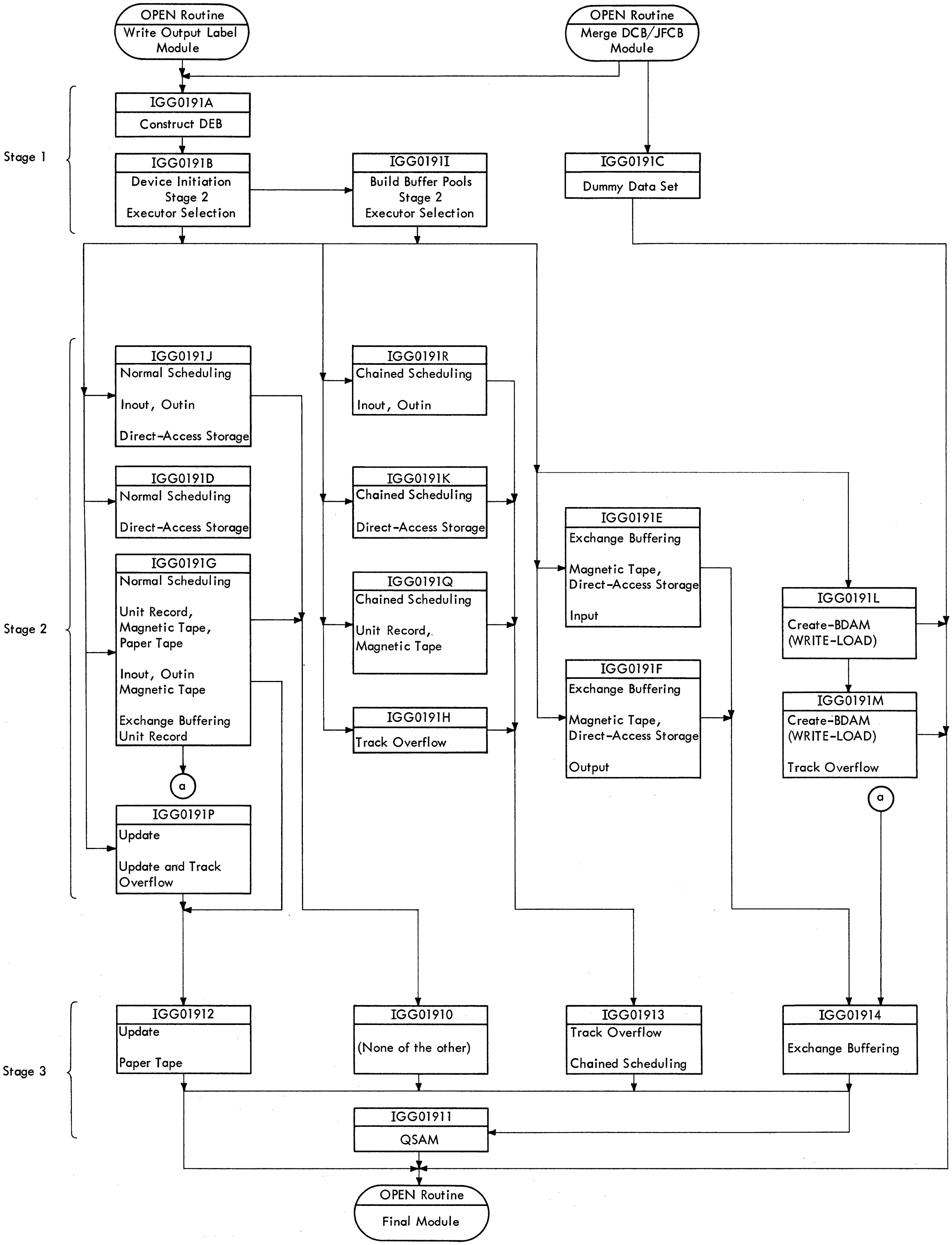


Figure 10. Flow of Control - SAM OPEN Executor

OPEN EXECUTORS

The OPEN executors are grouped into three stages. Those in the first stage receive control from the OPEN routine of I/O support. These executors pass control to one of the stage 2 executors, or return control to the OPEN routine. The stage 2 executors in turn, pass control to the stage 3 executors, or return control to the OPEN routine. Stage 3 executors return control to the OPEN routine. Before relinquishing control, each executor specifies the next executor to be called for the data set being opened, and also examines the where-to-go (WTG) table to determine whether other data sets being opened at the same time need its services. (For a description of the WTG table refer to the publication IBM System/360 Operating System: Input/Output Support, Program Logic Manual.)

Figure 10 shows the executors that compose the three stages, and their relationship.

STAGE 1 OPEN EXECUTORS

Stage 1 OPEN executors construct data extent blocks (DEB) and buffer pools. There are separate executors for actual data sets and for dummy data sets. The executor for actual data sets consists of three modules and passes control to a stage 2 executor (via an XCTL macro-instruction); the executor for dummy data sets consists of one module and returns control to the OPEN routine. Either executor receives control from the OPEN routine by being identified in the WTG table and being loaded into the supervisory transient area. On conclusion of all stage 1 executors' processing, the last enters in the WTG table the identification of the stage 2 executor that is required. Table 24 lists the access conditions that cause different stage 1 executors to be selected, loaded, and to receive control after loading.

Stage 1 OPEN Executor IGG0191A

Executor IGG0191A receives control from the OPEN routine, unless the DD statement is DUMMY. (If the DD statement is DUMMY, executor IGG0191C receives control from the OPEN routine.)

The executor operates as follows:

- It receives control after it is loaded.
- It computes the amount of main storage required for the data extent block (DEB), obtains the space, and enters the addresses of the extents. If no primary extent has been requested for an output data set, as shown by the value in the field DS1NOEPV in the data set control block (DSCB), the executor sets the DCBCIND1 field to show a volume-full condition.
- It specifies in the WTG table that executor IGG0191B is the next executor required for this DCB. It then searches the WTG table to pass control to another executor. For executor IGG0191A, this is always executor IGG0191B.

Table 24. OPEN Executor Selector - Stage 1 OPEN Executors

Access Conditions	Selection		
Actual data set	X	X	
Buffer Pool Required		X	
Dummy data set			X
Executors			
IGG0191A and IGG0191B	X	X	
IGG0191C			X
and IGG0191I		X	

Stage 1 OPEN Executor IGG0191B

Executor IGG0191B is always loaded after executor IGG0191A has completed processing all entries in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- If the device type is direct-access storage, it determines the first Seek address and enters it in the DCBFDAD field.
- If the DCB is opened for MOD, it copies the contents of the DS1TRBAL field of the DSCB into the DCBTRBAL field of the DCB.

- If the DCB is opened for input and the data set control block (DSCB) shows that the data set contains no data, it sets the DCBCIND1 field to show a volume-full condition. For example, for an error log data set without entries the DSCB field DS1LSTAR (which contains the value TTR) has an entry of TTR=0.

- If this or the preceding executor sets a volume-full indication in the DCB, the executor sets the IOBFLAG1 field (and the ICBFLAG1 field, if chained scheduling is used) to show an end-of-volume condition.

- If the device is a printer with the universal character set (UCS) feature, the executor constructs a channel program to prevent (block) or to allow (unblock) data checks for the printer, and issues an EXCP macro-instruction for it. (The IOB, DEB, and DCB located in the work area of the OPEN routine are used to schedule and execute the channel program.)

- If a buffer pool is to be built, as shown by entries in the DCBBUFNO or DCBBUFCB fields, the executor specifies in the WTG table that executor IGG0191I is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

- If no buffer pool is to be built, the executor specifies in the WTG table the stage 2 executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 1 OPEN Executor IGG0191C (and Dummy Data Set Module IGG019AV)

Executor IGG0191C operates as follows:

It receives control from the OPEN routine if the DD statement is DUMMY, and loads module IGG019AV. Dummy data sets require only this executor; if no other data sets are being opened, control returns to the OPEN routine.

Dummy data set module IGG019AV operates as follows:

It receives control when a sequential access method macro-instruction refers to a dummy data set. For a dummy input data set, the module passes control to the user's EODAD routine; for a dummy output data set, the module returns control to the processing program immediately, without scheduling any I/O operation.

Stage 1 OPEN Executor IGG0191I

Executor IGG0191I is loaded after executor IGG0191B if the OPEN executor must build buffer pools.

The executor operates as follows:

- It receives control after it is loaded.

- If the values in both the DCBBUFL and DCBBLKSI fields are zero, the executor passes control to the ABEND routine.

- If the value in either the DCBBUFL or DCBBLKSI field is not zero, the executor uses that value to establish the size of the buffer. The value in the field DCBBUFNO determines the number of buffers constructed.

- It specifies in the WTG table the stage 2 executor required for this DCB. It then searches the WTG table to pass control to another executor.

STAGE 2 OPEN EXECUTORS

A stage 2 OPEN executor establishes device oriented information for the processing described by a DCB, and completes device oriented control blocks or fields. One of the stage 2 executors receives control for each DCB being opened; the WTG table identifies the executor required for each DCB. On conclusion of an executor's processing it enters in the WTG table the identification of the stage 3 executor required. Table 25 lists the access conditions that cause the different stage 2 executors to be loaded and to receive control.

The device oriented processing performed by a stage 2 executor primarily consists of the construction of input/output blocks (IOB) and their associated channel programs, and the identification of the end-of-block routine required for the processing described by the DCB. For chained channel-program scheduling an executor also constructs interruption control blocks (ICB).

Table 25. OPEN Executor Selector - Stage 2 OPEN Executors

Access Conditions	Selection														
BSAM or	X^1				X^1	X	X^1	X	X^1	X	X	X^1	X^1	X^1	
QSAM	X^1	X	X	X	X^1		X^1		X^1			X^1	X^1	X^1	
Input or	X^2	X		X^2					X^2						
Output	X^2		X	X^2					X^2						
Inout, Outin						X		X							X
Update												X	X		
Unit Record				X	X^4									X^5	
Magnetic Tape		X^3	X^3		X^4	X								X^5	
Paper Tape					X^4										
Direct-Access Storage	X	X^3	X^3					X	X						
WRITE-LOAD (Create-BDAM)										X	X				
Exchange Buffering		X	X	X											
Track Overflow							X				X		X		
Chained Scheduling									X					X	X
Executors															
IGG0191D	D														
IGG0191E		E													
IGG0191F			F												
IGG0191G				G	G	G									
IGG0191H							H								
IGG0191J								J							
IGG0191K									K						
IGG0191L										L					
IGG0191M											M				
IGG0191P												P	P		
IGG0191Q														Q	
IGG0191R															R

[1] This executor is selected for either QSAM or BSAM.
[2] This executor is selected for either Input or Output.
[3] This executor is selected for either Magnetic Tape or Direct-Access Storage.
[4] This executor is selected for either Unit Record, Magnetic Tape, or Paper Tape.
[5] This executor is selected for either Unit Record or Magnetic Tape.

Stage 2 OPEN Executor IGG0191D

Executor IGG0191D receives control after executor IGG0191B or IGG0191I if the OPEN parameter list specifies:

- Input or Output

and the DCB specifies:

- Direct-access storage device

- BSAM, or QSAM and simple buffering

(but neither Update, nor track overflow, nor chained channel-program scheduling is specified). It may also receive control after executors IGG0191E, IGG0191F, or IGG0191K.

The executor constructs IOBs and channel programs and places the address of the first IOB into the DCB.

The executor specifies in the WTG table that executor IGG01910 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 OPEN Executor IGG0191E

Executor IGG0191E receives control after executor IGG0191B or IGG0191I if the OPEN parameter list specifies:

- Input

and the DCB specifies:

- Exchange buffering

- Magnetic tape, or direct-access storage

(but not track overflow). The executor is loaded, and gains control, when its identification in the WTG table is found by another executor.

The executor operates as follows:

- It receives control after it is loaded.

- If the operating mode is move, or the record format is variable-length blocked, or the record format is variable-length and the operating mode is substitute, simple buffering is substituted for exchange buffering. Therefore, it identifies (in the WTG table) executor IGG0191D (if the device type is direct-access storage) or executor IGG0191G (if the device-type is unit record) as the executor required next for this DCB. It then searches the WTG table to pass control to another executor.

- It identifies the end-of-block routine to be used in the processing specified by the DCB, and obtains space for and constructs IOBs and channel programs and links them.

- If the device is direct-access storage, it copies the starting Seek address from the DCB into the IOB.

- It specifies in the WTG table that executor IGG01914 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 OPEN Executor IGG0191F

Executor IGG0191F receives control after executor IGG0191I if the OPEN parameter list specifies:

- Output

and the DCB specifies:

- Exchange buffering

- Magnetic tape, or direct-access storage

(but not track overflow). The executor is loaded, and gains control, when its identification in the WTG table is found by another executor.

The executor operates as follows:

- It receives control after it is loaded.

- If the operating mode is move, or the record format is variable-length blocked, or the record-format is variable-length and the operating mode is substitute, simple buffering is substituted for exchange buffering. Therefore, it identifies (in the WTG table) executor IGG0191D (if the device type is direct-access storage) or IGG0191G (if the device type is unit record or magnetic tape) as the executor required next for this DCB. It then searches the WTG table to pass control to another executor.

- It identifies the end-of-block routine to be used in the processing specified by the DCB, and obtains space for and constructs IOBs and channel programs and links them.

- It specifies in the WTG table that executor IGG01914 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 OPEN Executor IGG0191G

Executor IGG0191G receives control after executor IGG0191B or IGG0191I if:

- The DCB specifies BSAM and either unit record, magnetic tape, or paper tape

- The DCB specifies QSAM, simple buffering, and either unit record, magnetic tape, or paper tape

- The DCB specifies QSAM, exchange buffering, and unit record

- The OPEN parameter is Inout or Outin and the DCB specifies magnetic tape

(but not if Update, track overflow, or chained channel-program scheduling is specified). It may also receive control after executors IGG0191E, IGG0191F, and IGG0191Q.

The executor constructs IOBs and channel programs and places the address of the first IOB into the DCB.

The executor specifies in the WTG table the next executor required for this DCB. If the DCB specifies exchange buffering, the next executor is IGG01914. If the DCB specifies paper tape, the next executor is IGG01912. For the remaining access conditions that cause this executor to be used, the next executor is IGG01910. The executor then searches the WTG table to pass control to another executor.

Stage 2 OPEN Executor IGG0191H

Stage 2 OPEN executor IGG0191H receives control after executor IGG0191B or IGG0191I, if the DCB specifies:

- Track overflow

(but not Update). The executor is loaded and gains control, when another executor finds its identification in the WTG table. (If both track overflow and Update are specified, executor IGG0191P receives control.)

The executor operates as follows:

- It receives control after it is loaded.

- It identifies the end-of-block routine and the direct-access NOTE/POINT routine to be used in the processing specified by this DCB.

- It obtains space for and constructs IOBs and channel programs for the maximum number of segments possible. It links the channel programs to the IOBs and the IOBs to one another.

- It specifies in the WTG table that executor IGG01913 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 OPEN Executor IGG0191J

Executor IGG0191J receives control after executor IGG0191B or IGG0191I if the OPEN parameter list specifies:

- Inout or Outin

and the DCB specifies:

- Direct-access storage.

The executor constructs IOBs and channel programs (including a portion for write-check, if it has been specified), and puts the address of the first IOB into the DCB.

The executor specifies in the WTG table that executor IGG01910 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 OPEN Executor IGG0191K

Executor IGG0191K receives control after executor IGG0191B or IGG0191I if the DCB specifies:

- Chained channel-program scheduling

- Direct-access storage.

It is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- If the NOTE/POINT macro-instruction is used, the executor identifies direct access NOTE/POINT module IGG019BK to be loaded for use with this DCB.
- It identifies the end-of-block routine to be loaded and used for the processing described by this DCB.
- It obtains space for, and constructs, one IOB, the required number of ICBs (that is, one ICB per channel program or buffer), and their associated channel programs, and then links them.
- It specifies in the WTG table that executor IGG01913 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 OPEN Executor IGG0191L

Executor IGG0191L receives control after executor IGG0191B or IGG0191I if the DCB specifies:

- Create-BDAM (WRITE-LOAD).

The executor constructs IOBs and enters the address of the first IOB into the DCB. If track overflow is not specified, the executor also builds channel programs. (If track overflow is specified, channel programs are built by executor IGG0191M.) This executor also loads the Create-BDAM WRITE and CHECK routines, and inserts their addresses into the DCB.

Unlike other stage 2 executors that cause control to pass to a stage 3 executor, this one indicates in the WTG table that OPEN executor processing for this DCB is completed, unless track overflow is specified. (If track overflow is specified, it identifies executor IGG0191M as the next executor required for this DCB.) It then searches the WTG table to pass control to another executor. If the WTG table has no other entries, the executor returns control to the OPEN routine.

Stage 2 OPEN Executor IGG0191M

Stage 2 OPEN executor IGG0191M constructs channel programs to write track overflow blocks using BSAM for a data set to be later processed by BDAM. Executor IGG0191L identifies it in the WTG table as its successor executor if the DCB specifies:

- Create-BDAM (WRITE-LOAD)
- Track overflow.

It is loaded and gains control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- If the extents are smaller than the blocks, it passes control to the ABEND routine.
- It constructs channel programs to write the number of segments required by the size of the block.
- It specifies in the WTG table that OPEN executor processing is completed for this DCB. It then searches the WTG table to pass control to another executor. If the WTG table has no other entries, the executor returns control to the OPEN routine.

Stage 2 OPEN Executor IGG0191P

Stage 2 OPEN executor IGG0191P receives control after executors IGG0191B or IGG0191I if the OPEN parameter list specifies:

- Update

(whether or not track overflow is also specified). It is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- It identifies module IGG019CC as the end-of-block routine to be loaded for use with the DCB.

- If the NOTE/POINT macro-instruction is specified, it identifies module IGG019BC as the NOTE/POINT routine to be loaded for use with this DCB.

- It obtains space for, and constructs, IOBs and channel programs to empty and refill each buffer. For QSAM, the executor links the channel programs so that a buffer may be either refilled only (by executing only the second half of the channel program) or may be emptied and refilled (by executing the channel program from the beginning).

- It specifies in the WTG table that executor IGG01912 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 OPEN Executor IGG0191Q

Executor IGG0191Q gains control after executors IGG0191B or IGG0191I if the DCB specifies:

- Chained channel-program scheduling

- Unit record, magnetic tape.

It is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.

- If the DCB specifies the CNTRL macro-instruction this executor identifies executor IGG0191G in the WTG table as the next executor to receive control for this DCB. It then searches the WTG table to pass control to another executor.

- If the NOTE/POINT macro-instruction is specified and the device is magnetic tape, it identifies module IGG019BL to be loaded for use with the DCB.

- If the NOTE/POINT macro-instruction is specified, and the device is unit record, it identifies dummy data set module IGG019AV to be loaded and used in place of NOTE/POINT.

- It identifies the end-of-block routine to be loaded and used for the processing described by this DCB.

- It obtains space for, and constructs, one IOB, the required number of ICBs (one per buffer or channel program), and channel programs appropriate to the device, and links them.

- It specifies in the WTG table that executor IGG01913 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 OPEN Executor IGG0191R

OPEN executor IGG0191R receives control after executors IGG0191B or IGG0191I if the OPEN parameter list specifies:

- Inout, or Outin

and the DCB specifies:

- Chained channel-program scheduling.

The executor is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.

- If the device is direct-access storage, it identifies NOTE/POINT module IGG019BK to be loaded for use with the DCB.

- If the device is magnetic tape, it identifies NOTE/POINT module IGG019BL to be loaded for use with the DCB.

- It identifies the end-of-block routine to be loaded for use with the DCB.

- It obtains space for, and constructs, one IOB, the required number of ICBs (one per buffer or channel program), and channel programs for direct-access storage or magnetic tape, and links them.

- It specifies in the WTG table that executor IGG01913 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

STAGE 3 OPEN EXECUTORS

A stage 3 executor identifies and loads the modules needed to perform the processing described by the DCB. If QSAM is used, and an input data set is to be processed, a second stage 3 executor also primes the buffers. Table 26 lists the access conditions that cause the different stage 3 executors to be loaded and to gain control.

Table 26. OPEN Executor Selector - Stage 3 OPEN Executors

Access Conditions	Selection						
Paper Tape	X						
Update		X					
Chained Scheduling			X				
Exchange Buffering				X			
Track Overflow					X		
None of the preceding						X	
QSAM							X
Executors							
IGG01910						X	
IGG01911							X
IGG01912	X	X					
IGG01913			X		X		
IGG01914				X			

Stage 3 OPEN Executor IGG01910

Executor IGG01910 receives control after executor IGG0191D or IGG0191J. It also receives control after executor IGG0191G unless the DCB specifies paper tape.

This executor operates as follows:

- It identifies and loads the device-independent routines.
- It loads the device-dependent routines identified by a mask set in stage 2.
- It enters the address of the routines into the DCB, and the address of appendages into the DEB appendage table.
- It enters the identification of each routine loaded, into the DEBSUBID field of the DEB.
- If QSAM is used, the executor specifies in the WTG table that executor IGG01911 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.
- If BSAM is used, the executor specifies in the WTG table that OPEN executor processing is completed for this DCB. It then searches the WTG table to pass control to another executor. If the WTG table has no other entries, the executor returns control to the OPEN routine.

Stage 3 OPEN Executor IGG01911

Executor IGG01911 is entered from executors IGG01910, IGG01912, IGG01913 and IGG01914 if the DCB specifies:

- GET, or PUT.

This executor operates as follows:

- It completes any remaining DCB fields.
- It completes the IOBs.
- For input it issues a BALR instruction to pass control to the end-of-block routine identified by a stage 2 executor and loaded by one of the other stage 3 executors. The end-of-block routine issues an EXCP macro-instruction to prime the buffers.
- It searches the WTG table to pass control to another executor. If the WTG table has no other entries, the executor returns control to the OPEN routine.

Stage 3 OPEN Executor IGG01912

Executor IGG01912 is entered from executor IGG0191P, and also from executor IGG0191G if the OPEN parameter is:

- Update

or if the DCB specifies:

- Paper Tape.

The executor operates as follows:

- It identifies and loads the device-independent routines.

- It loads the device-dependent routines.

- It enters the addresses of the routines into the DCB, and the address of the paper tape appendage into the appendage vector table.

- If QSAM is used, the executor specifies in the WTG table that executor IGG01911 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

- If BSAM is used, the executor specifies in the WTG table that OPEN executor processing is completed for this DCB. It then searches the WTG table to pass control to another executor. If the WTG table has no other entries, the executor returns control to the OPEN routine.

Stage 3 OPEN Executor IGG01913

Executor IGG01913 receives control after executor IGG0191H, IGG0191K, IGG0191Q, and IGG0191R, if the DCB specifies:

- Chained channel-program scheduling, or track overflow.

It is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.

- If QSAM is specified, it identifies, loads, and places the address into the DCB of:

  - A GET or a PUT routine
  - A synchronizing routine

  and specifies in the WTG table that executor IIG01911 is to receive control next for this DCB.

- If BSAM is specified, it identifies, loads, and places the address into the DCB of:

  - A READ or WRITE routine
  - A CHECK routine
  - A routine to serve the NOTE/POINT macro-instruction if it is specified

  and specifies in the WTG table that OPEN executor processing is completed for this DCB.

- It identifies and loads all the appendages required and places their addresses into the appendage vector table.

- It loads the end-of-block routine identified by a stage 2 executor and places its address into the DCB.

- It searches the WTG table to pass control to another executor. If the WTG table has no other entries, the executor returns control to the OPEN routine.

Stage 3 OPEN Executor IGG01914

Executor IGG01914 receives control after executor IGG0191E, IGG0191F, and IGG0191G if the DCB specifies:

- Exchange buffering.

It is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.

- If the access conditions specified are:

  Output and locate, or
  Input and move, or
  Input, locate, and variable-length

  it specifies in the WTG table that executor IGG01910 is required for this DCB.

  It then searches the WTG table to pass control to another executor.

- It identifies, loads, and puts the address into the DCB of:

  - A GET or a PUT routine
  - A synchronizing routine

  and specifies executor IGG01911 in the WTG table as the executor to receive control next for this DCB.

- It identifies and loads all the appendages required and places their addresses into the appendage vector table.

- It loads the end-of-block routine identified by a stage 2 executor and places its address into the DCB.

- It searches the WTG table to pass control to another executor.

CLOSE EXECUTORS

There are two CLOSE executors. The first one (IGG0201A) always receives control if one of the sequential access methods is used. The second one (IGG0201B) receives control after executor IGG0201A if QSAM was used with an output data set and a channel program encountered an error condition while executor IGG0201A had CPU control. Control returns to the CLOSE routine of I/O support when CLOSE executor processing is completed. Table 27 shows the conditions that cause the two executors to gain control.

Table 27. CLOSE Executor Selector

Access Conditions	Selection	
CLOSE macro-instruction	X	X
Permanent error or end-of-volume condition when using QSAM for output		X
Executors		
IGG0201A	X	X
IGG0201B		X

CLOSE EXECUTOR IGG0201A

Executor IGG0201A receives control from the CLOSE routine of I/O support if the DCBDSORG field specifies a value of PS or PO.

The executor operates as follows:

- It receives control after it is loaded.

- If both the OPEN parameter is Output and the DCB specifies PUT, the executor issues a TRUNC and a PUT macro-instruction to cause scheduling of the last buffer. On return of control, the executor awaits execution of the last channel program.

- If all channel programs were executed without encountering either an end-of-volume condition or a permanent error, the executor continues processing.

- If any of the preceding channel programs encountered either a permanent error or an end-of-volume condition, the executor specifies in the WTG table that executor IGG0201B is required for this DCB. Depending on the remaining entries in the WTG table, it then either processes another DCB, or passes control to executor IGG0201B.

- If either Output or PUT are not specified, the executor issues a PURGE macro-instruction for any pending channel programs. Note that when processing under BSAM the CHECK routine assures execution of all channel programs.

- If Output and either a DCBDSORG field value of PO, or WRITE or PUT with a DD statement of the form (MEMBERNAME) are specified, the executor issues a STOW macro-instruction. On completion of the STOW routine, the executor tests for I/O errors and for logical errors, such as insufficient space in the directory. For either type of error, the executor issues an ABEND macro-instruction with a code of hexadecimal 0B14.

- If QSAM and simple buffering are specified, the executor returns the buffers associated with the DCB to the buffer control block pointed to by the address in the field DCBBUFCB.

- The executor computes the amount of space occupied by the channel programs, IOBs (and ICBs, if chained scheduling is used), and returns that space to the supervisor via a FREEMAIN macro-instruction.

- The executor specifies in the WTG table that CLOSE executor processing is completed for this DCB. Depending on the remaining entries in the WTG table, it then either processes another DCB, passes control to executor IGG0201B, or returns control to the CLOSE routines.

CLOSE EXECUTOR IGG0201B (ERROR PROCESSING)

Executor IGG0201B receives control after executor IGG0201A if the latter finds that a channel program for an output data set using QSAM encountered a permanent error or an end-of-volume condition. It is loaded and receives control when its identification is found in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- It determines whether a channel program encountered a permanent error or an end-of-volume condition.
- If a channel program encountered a permanent error, the executor performs its remaining processing. Any buffers not written out are not processed.
- If a channel program encountered an end-of-volume condition, the executor finds the IOB associated with that channel program and places its address into the DCBIOBA field. It then passes control to the Output synchronizing routine for normal processing of the end-of-volume condition. When control returns, the executor performs its remaining processing, unless one of the channel programs encountered a permanent error or another end-of-volume condition. In either of those cases, it resumes processing as when it first received control.
- If Output and either a DCBDSORG field value of PO, or WRITE or PUT with a DD statement of the form (MEMBERNAME) are specified, the executor issues a STOW macro-instruction. On completion of the STOW routine, the executor tests for I/O errors and for logical errors, such as insufficient space in the directory. For either type of error, the executor issues an ABEND macro-instruction with a code of hexadecimal 0B14.
- If QSAM and simple buffering are specified, the executor returns the buffers associated with the DCB to the buffer control block pointed to by the address in the field DCBBUFCB.
- The executor computes the amount of space occupied by the channel programs, IOBs (and ICBs, if chained scheduling is used), and returns that space to the supervisor via a FREEMAIN macro-instruction.
- The executor specifies in the WTG table that CLOSE executor processing is completed for this DCB. Depending on the remaining entries in the WTG table, the executor either processes another DCB or returns control to the CLOSE routine.

SYNAD/EOV EXECUTOR IGC0005E (SVC 55)

Executor IGC0005E performs error-condition processing. If a synchronizing and error routine (in QSAM), or a CHECK routine (in BSAM), finds that the execution of a channel program encountered either a permanent error or an end-of-volume (EOV) condition, the routine issues an SVC 55 instruction. (The Update Synchronizing and Error Processing routine passes control to this executor only for an end-of-volume condition; the Paper Tape Synchronizing and Error Processing routine never passes control to this executor.) An SVC 55 instruction causes this executor to be loaded and to receive control.

Control passes to and from this executor along three paths, depending upon whether control was received due to an EOV condition, due to a permanent error condition and there is a SYNAD routine present, or due to a permanent error condition and there is no SYNAD routine present. The flow of control under these three conditions in QSAM is shown in Figure 11, for BSAM, it is shown in Figure 12.

For an EOV condition, the executor operates as follows:

- It obtains a work area.
- It passes control to the end-of-volume routine of I/O support. If that routine finds a new volume, it eventually passes control to EOV/new volume executor. After processing, the executor returns control to the synchronizing and error processing or to the CHECK routine.

If there is no SYNAD routine present, the executor operates as follows for a permanent error condition:

- For QSAM, the executor implements the error options specified in the field DCBEROPT in the DCB. It returns control to the synchronizing routine for the SKIP or ACCEPT option.
- For BSAM, the executor passes control to the ABEND routine.

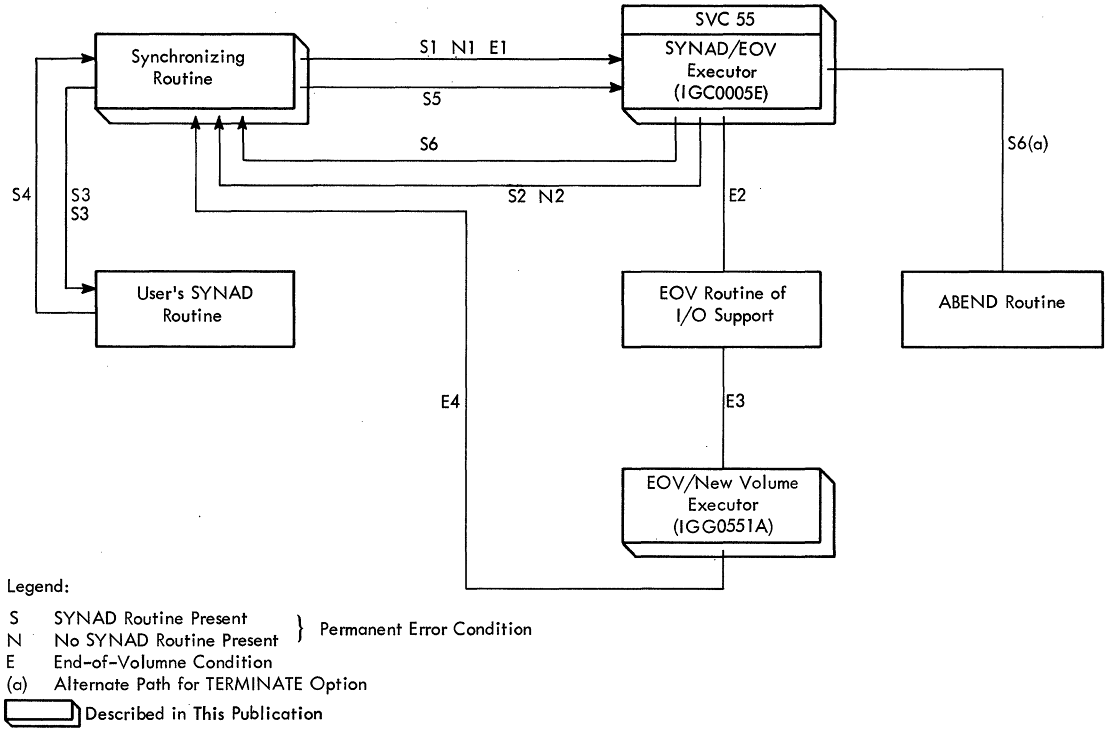

SVC 55
SYNAD/EOV Executor (IGC0005E)
Synchronizing Routine
S1 N1 E1
S5
S6
S6(a)
S4
S3
S3
S2 N2
E2
User's SYNAD Routine
EOV Routine of I/O Support
ABEND Routine
E4
E3
EOV/New Volume Executor (IGG0551A)
Legend:
S SYNAD Routine Present
N No SYNAD Routine Present
} Permanent Error Condition
E End-of-Volumne Condition
(a) Alternate Path for TERMINATE Option
Described in This Publication

Figure 11. Flow of Control To and From the SYNAD/EOV Executor (IGC0005E) in QSAM

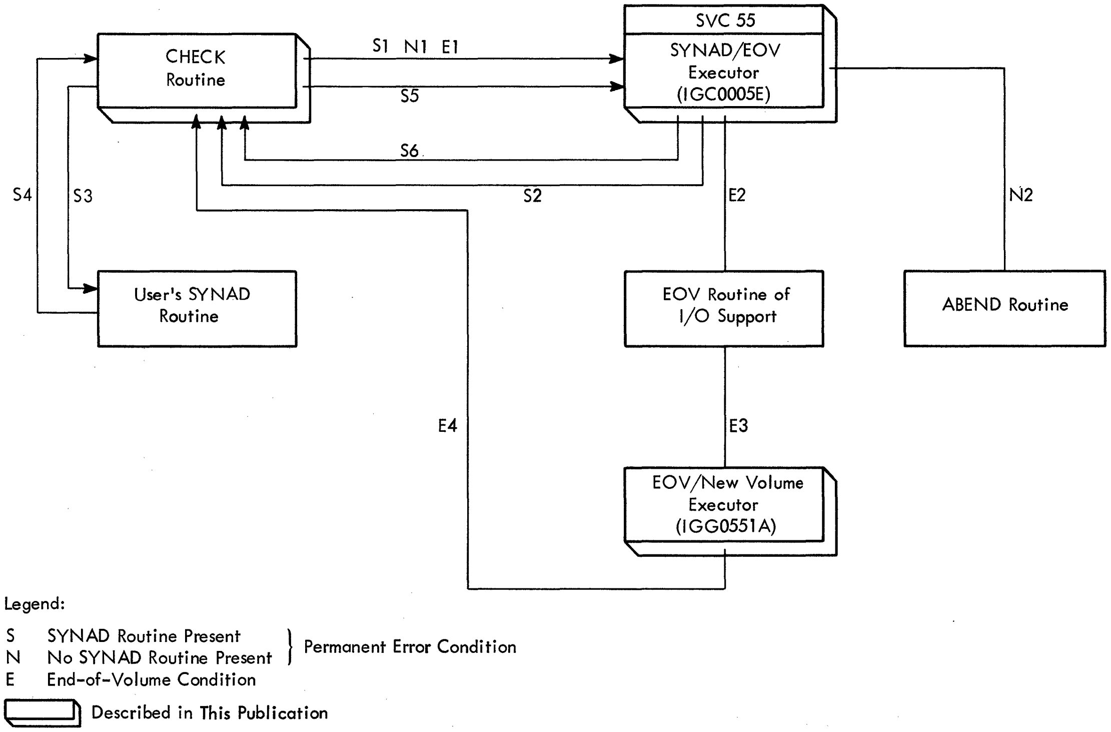

SVC 55
SYNAD/EOV Executor (IGC0005E)
CHECK Routine
S1 N1 E1
S5
S6
S4
S3
S2
E2
N2
User's SYNAD Routine
EOV Routine of I/O Support
ABEND Routine
E4
E3
EOV/New Volume Executor (IGG0551A)
Legend:
S SYNAD Routine Present
N No SYNAD Routine Present
} Permanent Error Condition
E End-of-Volume Condition
Described in This Publication

Figure 12. Flow of Control To and From the SYNAD/EOV Executor (IGC0005E) in BSAM

If there is a SYNAD routine present, the executor operates as follows for a permanent error condition:

- For QSAM, the executor returns control to the synchronizing routine. (The synchronizing routine then passes control to the user's SYNAD routine. After error processing, the user's SYNAD routine may return control to the synchronizing routine. The synchronizing routine issues a second SVC 55 instruction to pass control to this executor.)

- For QSAM, the executor then implements the error option and returns control to the synchronizing routine (for the SKIP or ACCEPT option).

- For BSAM, the executor returns control to the CHECK routine. (The CHECK routine passes control to the user's SYNAD routine. A return of control from the SYNAD routine to the CHECK routine in BSAM is interpreted as an ACCEPT error option. The CHECK routine issues a second SVC 55 instruction to pass control to this executor again.)

- For BSAM, the executor then implements the ACCEPT error option and returns control to the processing program.

The executor implements error options in the following manner:

- For the TERMINATE error option, the executor passes control to the ABEND routine.

- For the ACCEPT error option, the executor issues EXCP macro-instructions to reschedule all channel programs except the one executed with an error. If the device is a printer all channel programs are rescheduled.

- For the SKIP error option, the executor issues EXCP macro-instructions to reschedule all channel programs, including the one executed with an error.

FEOV EXECUTOR IGC0003A (SVC 31)

Executor IGC0003A causes reading or writing to be discontinued for the balance of the present volume and permits the processing program to start reading or writing a new volume. The FEOV (force-end-of-volume) macro-expansion includes an SVC 31 instruction that causes this executor to be loaded and to gain control.

For an input data set, processed under QSAM or BSAM, the executor operates as follows:

- It receives control when the processing program uses an FEOV macro-instruction.

- It obtains a work area by means of a GETMAIN macro-instruction.

- It prevents the execution of any pending channel programs by means of the PURGE macro-instruction.

- It passes control, and the work area, to the end-of-volume routine of I/O support by means of an XCTL macro-instruction.

For an output data set processed under BSAM, the executor operates as follows:

- It receives control when the processing program uses an FEOV macro-instruction.

- It obtains a work area by means of a GETMAIN macro-instruction.

- It passes control, and the work area, to the end-of-volume routine of I/O support by means of an XCTL macro-instruction.

For an output data set processed under QSAM, the operation of the executor, and the resultant flow of control, depends on the operating mode and how certain channel programs execute. The operation and flow of control for each possible combination of mode and channel program execution is described in detail in Appendix E.

In general, assuming normal execution of all channel programs, the executor operates as follows:

- It receives control when the processing program uses an FEOV macro-instruction.

- It obtains a work area by means of a GETMAIN macro-instruction.

- It passes control to the PUT routine to cause scheduling of the present buffer for output.

- It awaits execution of all pending channel programs.

- It passes control, and the work area, to the end-of-volume routine of I/O support by means of an XCTL macro-instruction.

EOV/NEW VOLUME EXECUTOR IGG0551A

Executor IGG0551A schedules, for execution with the new volume, any channel programs not executed with the old volume. The end-of-volume routine of I/O support issues an XCTL macro-instruction to pass control to this executor after the routine has caused the mounting of the next volume of the input data set; for an output data set, the routine passes control to this executor after the routine has mounted a new volume, or acquired additional space on the current volume.

The executor operates as follows:

- It receives control when the next, new, or more volume is available.
- It resets all indications of the end-of-volume condition in the DCB.
- If the device type is direct-access, the executor inserts the new full device address (FDAD) into the DCB and the IOB.
- It issues BALR instructions to pass pending channel programs to the end-of-block routine to have them scheduled for execution. (If Create-BDAM - WRITE-LOAD is specified, control passes to the Create-BDAM WRITE routine.)
- It issues a FREEMAIN macro-instruction for the work area obtained for the end-of-volume routine.
- It returns control to the routine that passed control to the end-of-volume routine via the SVC 55 instruction. For a normal end-of-volume condition found by a synchronizing or CHECK routine, control returns to the synchronizing or CHECK routine. For a forced end-of-volume condition established by an FEOV macro-instruction in the processing program, control returns to the processing program. For an end-of-volume condition arising during the FEOV executor, control returns to the FEOV executor.

BUFFER POOL MANAGEMENT

Buffer pool management routines form main storage space into buffers, and they return buffers that are no longer needed. There are five buffer pool management routines:

- GETPOOL - This routine obtains main storage and forms a buffer pool.
- BUILD - This routine forms a buffer pool in main storage supplied by the processing program.
- GETBUF - This routine provides buffers from the buffer chain.
- FREEBUF - This routine returns buffers to the buffer pool.
- FREEPOOL - This routine returns main storage previously used for a buffer pool.

GETPOOL MODULE IECQBFG1

Module IECQBFG1 obtains main storage space and forms it into buffers. It is loaded at execution time by a LINK macro-instruction.

The module operates as follows:

- It rounds the buffer length to the next higher double-word multiple if the specified length is not such a multiple.
- It determines buffer alignment from the DCBBUFAL field value in the DCB.
- It computes the number of bytes required and issues a GETMAIN macro-instruction.
- It constructs a buffer pool control block in the first eight bytes of storage obtained.
- If double-word (not-full-word) alignment is specified in the DCBBUFAL field in the DCB, the module starts the first buffer at the byte immediately following the BUFCB.
- If full-word (not-double-word) alignment is specified in the DCBBUFAL field, the module skips one word after the buffer pool control block before starting the first buffer.
- It chains the first buffer to the buffer pool control block and determines the start of the next buffer by adding the rounded buffer length value tc the address of the first buffer. The module chains the next buffer to the preceding buffer, and continues until all the buffers have been chained.
- It returns control to the processing program.

Figure 13 illustrates the buffer pool control block (BUFCB) that describes the buffer pcol. Figure 14 illustrates the buffer pool structures formed by the GETPCOL module.

	BUFAD	BUFNO	BUFL	
BUFCB	Address of First Available Buffer	Number of Buffers Requested	Length of Each Buffer	
Byte	0	4	6	8

Figure 13. Buffer Pool Control Block

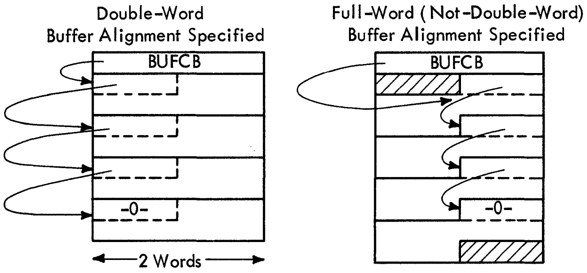

Double-Word
Buffer Alignment Specified
BUFCB
-0-
2 Words
Full-Word (Not-Double-Word)
Buffer Alignment Specified
BUFCB
-0-

Figure 14. GETPOOL Buffer Pool Structures

BUILD MODULE IECBBFB1

Module IECBBFB1 forms main storage space supplied by the processing program into buffers. It is loaded at execution time by a LINK macro-instruction.

The module operates as follows:

- It rounds the buffer length to the next higher full-word multiple if the specified length is not such a multiple.

- It constructs a buffer pool control block in the first eight bytes of the main storage space provided by the processing program.

- It starts the first buffer at the byte immediately following the buffer pool control block.

- It chains the first buffer to the buffer pool control block and determines the start of the next buffer by adding the rounded buffer length value to the address of the first buffer. The module chains the next buffer to the preceding buffer, and continues until all the buffers are chained.

- It returns control to the processing program.

Table 28 lists for each possible combination of space alignment and buffer length parity the illustration that shows the structure of the resulting buffer chain or pool. Figure 13 illustrates the buffer pool control block (BUFCB), Figure 15 illustrates the various buffer alignments that the BUILD module forms.

GETBUF MACRO-EXPANSION

The purpose of this coding is to provide the next buffer from the buffer pool. The macro-expansion produces in-line code that presents the address of the next buffer to the processing program and updates the buffer pool control block to point at the following buffer.

FREEBUF MACRO-EXPANSION

The purpose of this coding is to return a buffer to the buffer chain. The macro-expansion produces in-line code that stores the address presently in the buffer pool control block in the first word of the buffer being returned, and then stores the address of that buffer in the buffer pool control block.

Table 28. BUILD Buffer Structuring Table

Alignment of First Byte of Space Passed in BUILD Macro-Instruction	Parity of Number of Words in Buffer Length after Rounding Up Length Parameter of BUILD Macro-Instruction	Buffer Pool Structure
Double - Word	Even	A
	Odd	B
Full - Word (Not - Double - Word)	Even	C
	Odd	D

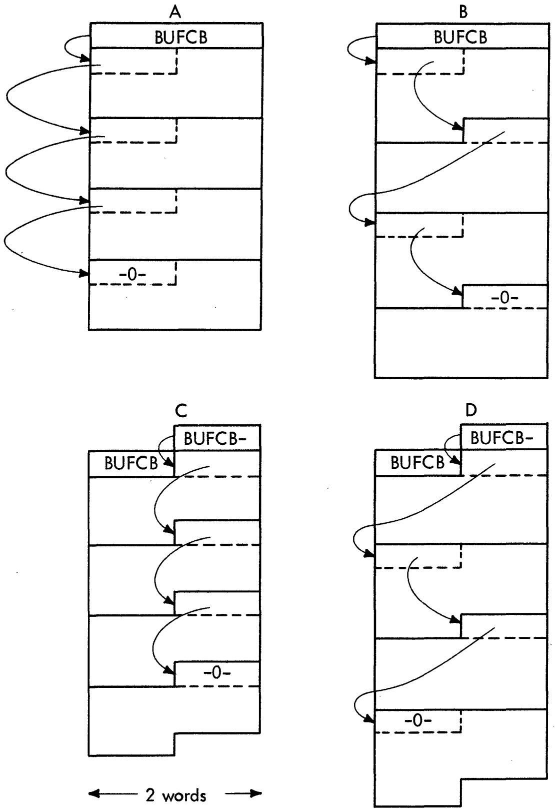


Figure 15. BUILD Buffer Pool Structures

FREEPOOL MACRO-EXPANSION

The purpose of this coding is to return the space previously allotted to the buffer chain to available main storage. The macro-expansion produces in-line code that computes the total number of bytes to be returned, issues a FREEMAIN macro-instruction, and sets the DCBBUFCB field in the DCB to show that no buffer pool is associated with that DCB.

These decision tables show the routines available and the access conditions that cause a routine to be used. They duplicate the decision tables in the text in table number, form, and content. A table that occupies a whole page may be out of sequence.

Table 2. Module Selector - Simple Buffering GET Modules

Access Conditions	Selections													
INPUT, GET, Simple Buffering	X	X	X	X	X	X	X	X	X					X
RDBACK, GET, Simple Buffering										X	X	X	X	
Locate operating mode	X	X	X							X	X			
Move operating mode				X	X	X	X	X	X			X	X	X
Fixed-length record format	X			X			X			X		X		
Undefined-length record format		X			X			X			X		X	
Variable-length record format			X			X			X					
Card reader, only a single buffer, CNTRL							X	X	X					
Character conversion for paper tape														X
GET Modules														
IGG019AA	X	X												
IGG019AB			X											
IGG019AC				X	X									
IGG019AD						X								
IGG019AG							X	X						
IGG019AH									X					
IGG019AM										X	X			
IGG019AN												X	X	
IGG019AT[1]														X

[1] This module also includes the Paper Tape Character Conversion Synchronizing and Error Processing routine.

Table 3. Module Selector - Exchange Buffering GET Modules

Access Conditions	Selections						
Input, GET, Exchange	X	X	X	X	X	X	X
Locate	X	X	X	X			
Substitute					X	X	X
Fixed-length	X	X			X		X
Variable-length			X				
Undefined-length				X		X	
Unblocked		X	X	X	X	X	
Blocked	X						X
GET Modules							
IGG019EA	X						
IGG019EB		X	X	X			
IGG019EC					X	X	
IGG019ED							X

Table 4. Module Selector - Update Mode GET Module

Access Conditions	Selections				
Update, GET	X	X	X	X	X
Fixed-length record format	X	X			
Variable-length record format			X	X	
Undefined-length record format					X
Blocked record format	X		X		
Unblocked record format		X		X	X
GET Module					
IGG019AE[1]	X	X	X	X	X

[1] This module also carries the Update mode PUTX routine

Table 5. Module Selector - Simple Buffering PUT Modules

Access Conditions	Selections					
Output, PUT/PUTX, Simple buffering	X	X	X	X	X	X
Locate operating mode	X	X	X			
Move operating mode				X	X	X
Fixed-length record format	X			X		
Undefined-length record format		X			X	
Variable-length record format			X			X
PUT Modules						
IGG019AI	X	X				
IGG019AJ			X			
IGG019AK				X	X	
IGG019AL						X

Table 6. Module Selector - Exchange Buffering PUT Modules

Access Conditions	Selections						
Output, PUT/PUTX Exchange	X	X	X	X	X	X	X
Move mode	X	X	X			X	
Substitute mode				X	X		X
Unblocked record format	X	X	X	X	X		
Blocked record format						X	X
Fixed-length record format	X			X		X	X
Variable-length record format		X					
Undefined-length record format			X		X		
PUT Modules							
IGG019EE	X	X	X	X	X		
IGG019EF						X	X

Table 7. Module Selector - Ordinary End-of-Block Modules

Access Conditions	Selections									
Normal channel-program scheduling	X	X	X	X	X	X	X	X	X	X
Input, or				X	X	X				
Update			X							
Output, or							X			
Inout,Outin				X		X	X			
Card reader or paper tape reader	X									
Printer or card punch								X	X	X
Magnetic tape		X								
Direct-access storage				X		X	X			
Track overflow					X					
Record format is not fixed-length standard				X						
Record format is fixed-length standard						X				
No control character								X		
Machine control character									X	
ASA control character										X
PRTOV-No user exit								X	X	X
End-of-Block Modules										
IGG019CC	X	X	X	X	X					
IGG019CD						X	X			
IGG019CE								X	X	
IGG019CF										X

Table 9. Module Selector - Chained Channel-Program Scheduling End-of-Block Modules

Access Conditions	Selections							
Chained channel-program scheduling	X	X	X	X	X	X	X	X
Input	X	X		X				
Output			X		X	X	X	X
Card reader	X							
Printer or card punch						X	X	X
Magnetic tape		X	X					
Direct-access storage				X	X			
No control character						X		
Machine control character							X	
ASA control character								X
End-of-Block Modules								
IGG019CV					X			
IGG019CW	X	X	X	X				
IGG019CX						X	X	
IGG019CY								X

Table 10. Module Selector - Track Overflow End-of-Block Module

Access Conditions	Selections
Output, Inout, Outin	X
Track Overflow	X
End-cf-Block Module	
IGG019C2	X

Table 11. Module Selector - Synchronizing and Error Processing Modules

Access Conditions	Selections			
GET	X	X		X
PUT			X	
Input, Readback		X		
Output			X	
Update	X			
Paper tape character conversion				X
Modules				
IGG019AF	X			
IGG019AQ		X		
IGG019AR			X	
IGG019AT[1]				X

[1]This module includes both the paper tape synchronizing and error processing routine and the paper tape GET routine. Both routines are described in the GET routines section of this publication.

Table 12. Module Selector - Track Overflow Asynchronous Error Processing Module

Access Conditions	Selections	
GET	X	
READ		X
Input, Inout, Outin	X	X
Track Overflow	X	X
Module		
IGG019C1	X	X

Table 14. Module Selector - Control Modules

Access Conditions	Selections	
CNTRL	X	X
Printer	X	
Card Reader, a single buffer		X
Modules		
IGG019CA	X	
IGG019CB		X

Table 15. Control Routines That Are Macro-Expansions

Macro-Instruction	Number of Macro-Expansions
PRTOV - User exit	1
PRTOV - No user exit	1

Table 13. Module Selector - Appendages

Access Conditions	Selections											
Input, Inout, Outin				X	X	X						
Readback					X							
Update	X	X	X									
Sysin							X					
GET	X											
READ		X										
Record format is fixed-length									X			
Record format is fixed-length blocked					X							
Record format is variable-length						X						
Record format is not fixed-length standard				X								
Direct-access storage				X							X	
Printer								X				
Paper tape									X			
Chained scheduling										X	X	
Track overflow												X
Appendages												
IGG019AW	AW											
IGG019BM		BM										
IGG019CG			CG									
IGG019CH				CH								
IGG019CI					CI							
IGG019CJ						CJ						
IGG019CK							CK					
IGG019CL								CL				
IGG019CS									CS			
IGG019CU										CU		
IGG019CZ											CZ	
IGG019C3												C3
Exits												
End-of-Extent	AW	BM		CH							CZ	
SIO			CG					CL				
Channel End					CI	CJ	CK		CS	CU		
PCI										CU		
Abnormal End										CU		C3

Table 17. Module Selector - READ and WRITE Modules

Access Conditions	Selections							
Input, or	X		X	X				
Output, or		X				X	X	X
Input, Outin	X	X						
Update					X			
READ	X		X	X	X			
WRITE		X						
WRITE (LOAD) (Create-BDAM)						X	X	X
Paper tape character conversion			X	X				
Fixed-length record format			X			X		X
Undefined-length record format or				X			X	
Variable-length record format							X	
Track Overflow								X
READ,WRITE Modules								
IGG019BA	X	X						
IGG019BF			X	X				
IGG019BH					X			
IGG019DA						X		
IGG019DB							X	
IGG019DD								X

Table 18. Routine Selector - CHECK Routines

Access Conditions	Selections				
Input or	X		X		
Output or		X			
Inout, Outin	X	X			
Update				X	
READ	X		X		
WRITE		X			
WRITE (LCAD) (Create-BDAM)					X
Paper tape character conversion			X		
CHECK Modules					
IGG019BB	X	X			
IGG019BG			X		
IGG019BI				X	
IGG019DC					X

Table 19. Module Selector - Control Modules Selected and Loaded by the OPEN Executor

Access Conditions	Selection						
NOTE/POINT	X	X		X	X		
Update, Track Overflow, or				X			
Chained Scheduling				X	X		
CNTRL			X			X	X
Direct-Access Storage	X			X			
Magnetic Tape		X	X		X		
Printer						X	
Card Reader							X
Control Modules							
IGG019BC	X						
IGG019BD		X					
IGG019BE			X				
IGG019BK				X			
IGG019BL					X		
IGG019CA[1]						X	
IGG019CB[1]							X

[1] These routines are also used in QSAM; see that section for description of these routines.

Table 20. Control Modules Loaded at Execution Time

SVC No.	Macro-Instruction	Function	Module No.
25	(none)	Establish valid track balance Erase balance of extent for track overflow	IGC0002E
69	BSP	Device Independent Backspace (tape, direct-access)	IGC0006I

Table 21. Control Routines That Are Macro-Expansions[1] [2]

Macro-Instruction	Number of Macro Expansions
PRTOV - User exit	1
PRTOV - No user exit	1

[1] These routines are also used in QSAM; see that section for a description of the routines.

[2] This table duplicates Table 15; it is repeated here to identify all control routines available in BSAM.

Table 22. BPAM Routines Residence

BPAM Routines	Module Number	Residence	Instruction Passing Control
STOW	IGC0002A	Supervisory Transient Area	SVC 21
FIND (C Option)	(Macro Expansion)	Processing Program Area	FIND (C Option)
FIND (D Option)	IECPFIND,IECPFND1	Supervisory Resident Area	SVC 18
BLDL	IECPFIND,IECPFND1	Supervisory Resident Area	SVC 18 or BAL IECPBLDL
Convert TTR	IECPFIND,IECPFND1	Supervisory Resident Area	BAL IECPCNVT
Convert MBBCCHHR	IECPFIND,IECPFND1	Supervisory Resident Area	BAL IECPRLTV

Table 23. Sequential Access Method Executors - Control Sequence

Executor	Number	Receives Control From	Via	Passes Control To
OPEN	See Tables 24, 25, 26	See Figure 10	XCTL (WTG Table)	See Figure 10
CLOSE	IGG0201A IGG0201B	CLOSE Routine	XCTL (WTG Table)	CLOSE Routine
SYNAD/EOV	IGC0005E	Synchronizing, CHECK Routines	SVC 55	EOV Routine
FEOV	IGC0003A	Processing Program	FEOV Macro-Instruction (SVC 31)	EOV Routine
EOV/new volume	IGG0551A	EOV Routine	XCTL	See Executor Description

Table 24. OPEN Executor Selector - Stage 1 OPEN Executors OPEN Executors

Access Conditions	Selection		
Actual data set	X	X	
Buffer Pool Required		X	
Dummy data set			X
Executors			
IGG0191A and IGG0191B	X	X	
IGG0191C			X
and IGG0191I		X	

Table 26. OPEN Executor Selector - Stage 3 OPEN Executors

Access Conditions	Selection						
Paper Tape	X						
Update		X					
Chained Scheduling			X				
Exchange Buffering				X			
Track Overflow					X		
None of the preceding						X	
QSAM							X
Executors							
IGG01910						X	
IGG01911							X
IGG01912	X	X					
IGG01913			X		X		
IGG01914				X			

Table 25. OPEN Executor Selector - Stage 2 OPEN Executors

Access Conditions	Selection														
BSAM or	X^1				X^1	X	X^1	X	X^1	X	X	X^1	X^1	X^1	
QSAM	X^1	X	X	X	X^1		X^1		X^1			X^1	X^1	X^1	
Input or	X^2	X		X^2					X^2						
Output	X^2		X	X^2					X^2						
Inout, Outin						X		X							X
Update												X	X		
Unit Record				X	X^4									X^5	
Magnetic Tape		X^3	X^3		X^4	X								X^5	
Paper Tape					X^4										
Direct-Access Storage	X	X^3	X^3					X	X						
WRITE-LOAD (Create-BDAM)										X	X				
Exchange Buffering		X	X	X											
Track Overflow							X				X		X		
Chained Scheduling									X					X	X
Executors															
IGG0191D	D														
IGG0191E		E													
IGG0191F			F												
IGG0191G				G	G	G									
IGG0191H							H								
IGG0191J								J							
IGG0191K									K						
IGG0191L										L					
IGG0191M											M				
IGG0191P												P	P		
IGG0191Q														Q	
IGG0191R															R

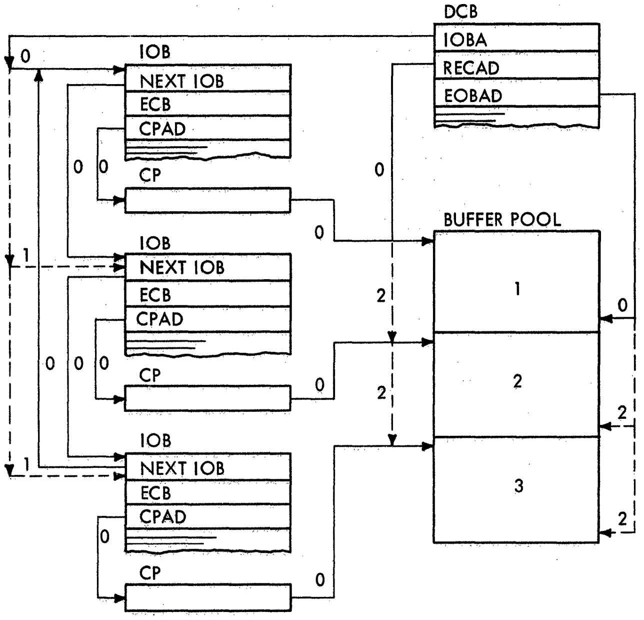
[1] This executor is selected for either QSAM or BSAM.
[2] This executor is selected for either Input or Output.
[3] This executor is selected for either Magnetic Tape or Direct-Access Storage.
[4] This executor is selected for either Unit Record, Magnetic Tape, or Paper Tape.
[5] This executor is selected for either Unit Record or Magnetic Tape.

Table 27. CLOSE Executor Selector

Access Conditions	Selection	
CLOSE macro-instruction	X	X
Permanent error or end-of-volume condition when using QSAM for output		X
Executors		
IGG0201A	X	X
and IGG0201B		X

APPENDIX B: QSAM CONTROL BLOCKS

Figure 16 shows the control blocks used in QSAM. Through the data control block (DCB), the QSAM routines associate the data being processed with the processing program. Fields in the DCB point to the start of a buffer, the end of a buffer, and an input/output block (IOB). These fields are updated as successive channel programs are executed. Each IOB points at the next IOB and at a channel program (CP), and carries an event control block (ECB) that the I/O supervisor posts after the channel program has been executed.

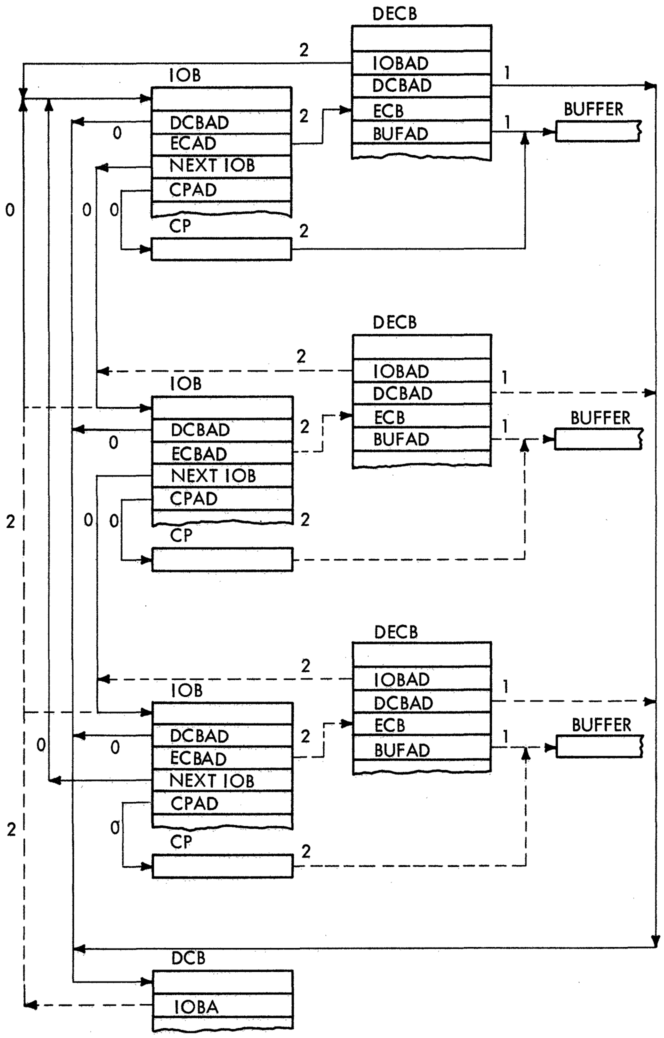


Legend:
Address Values:
0 Entered by the OPEN executor.
1 Updated by the synchronizing routine.
2 Updated by the GET or PUT routine.
– – – Successive Address Values

Figure 16. QSAM Control Blocks

Figure 17 shows the control blocks used in BSAM and their stages of completion. Stage 0 shows the state of the control blocks before any READ or WRITE macro-instruction. Stage 1 shows the effect of the READ or WRITE macro-instruction, that is, the values supplied by the processing program in the data event control block (DECB). Finally, stage 2 shows the effect of the READ or WRITE routine's tying together these control blocks.

Before any READ or WRITE macro-instruction, the data control block (DCB) points to the first input/output block (IOB). This IOB points back to the DCB, to the next IOB, and to the channel prcgram (CP). The READ or WRITE macro-instruction identifies the DCB and the buffer to be read into or written out. Finally, the READ or WRITE routine connects the DECB with the current IOB, inserts the address of the ECB (which is located in the DECB) into the IOB, and points the channel program to the buffer. Successive macro-instructions cause updating of the IOB address in the DCB and insert address values in the next DECB, IOB, and channel program.



Legend:
Address Values
0 Entered by the OPEN Executor.
1 Provided by the processing program.
2 Completed by the READ or WRITE routine.
--- Successive Address Values.

Figure 17. BSAM Control Blocks

APPENDIX D: PAPER TAPE CODE CONVERSION - DESCRIPTION OF TABLE MODULES

GET routine IGG019AT (paper tape) and WRITE routine IGG019BF (paper tape) use the tables in the following modules to convert characters read from paper tape to EBCDIC characters.

CODE CONVERSION MODULE IGG019CM

This module is loaded by the OPEN executor if the DCB specifies paper tape, and code conversion for teletype transmission code.

The module consists of three tables:

- A validity checking and special functions table.
- A lower case character translation table.
- An upper case character translation table.

CODE CONVERSION MODULE IGG019CN

This module is loaded by the OPEN executor if the DCB specifies paper tape, and code conversion for ASCII paper tape code.

The module consists of two tables:

- A validity checking and special functions table.
- A character translation table.

CODE CONVERSION MODULE IGG019CO

This module is loaded by the OPEN executor if the DCB specifies paper tape, and code conversion for Burroughs paper tape code.

The module consists of two tables:

- A validity checking and special functions table.
- A character translation table.

CODE CONVERSION MODULE IGG019CP

This module is loaded by the OPEN executor if the DCB specifies paper tape, type and code conversion for Friden paper tape code.

The module consists of three tables:

- A validity checking and special functions table.
- A lower case character translation table.
- An upper case character translation table.

CODE CONVERSION MODULE IGG019CQ

This module is loaded by the OPEN executor if the DCB specifies paper tape, and code conversion for IBM PTTC/8 code.

The module consists of three tables:

- A validity and special functions table.
- A lower case character translation table.
- An upper case character translation table.

CODE CONVERSION MODULE IGG019CR

This module is loaded by the OPEN executor if the DCB specifies paper tape, and code conversion for NCR paper tape code.

The module consists of three tables:

- A validity checking and special functions table.
- A lower case character translation table.
- An upper case character translation table.

The operation of the FEOV executor for an output data set processed under the queued sequential access method (QSAM) depends on the operating mode and the execution of certain channel programs.

In the move operating mode, the execution of all channel programs is tested by the FEOV executor. It awaits the execution of the channel program for the present buffer, and causes processing of any error conditions.

In the locate operating mode, the execution of the channel program for the next buffer in the chain is tested by the Output synchronizing routine. This test occurs immediately after the end-of-block routine has caused the channel program for the present buffer to be scheduled for execution. The execution of the channel programs for all the following buffers, including the one just scheduled, is tested by the FEOV executor after the last channel program has executed.

When a QSAM routine tests the execution of a channel program, one of three conditions may be established, with the stated results:

- The channel program executed normally: Normal processing continues.
- The channel program is not yet executed: The testing routine awaits completion of the channel program.
- The channel program executed with an error condition: The testing routine passes control to the SYNAD/EOV executor (IGC0005E), by means of an SVC 55 instruction in synchronizing routine IGG019AR. The executor distinguishes between permanent error conditions and end-of-volume conditions. (For a description of the error processing operations initiated by the SYNAD/EOV executor, refer to the section: Sequential Access Method Executors, in this publication.)

The FEOV executor substitutes its own SYNAD routine (contained within module IGC0003A) for the processing program's. That SYNAD routine releases the work area normally obtained by the executor and issues an ABEND macro-instruction.

The cperation of the FEOV executor, and the resultant flow of control between it and other control program routines, differs for each of eight conditions. The conditions are described below. Figure 18 illustrates the flow of control between the executor and other routines. Table 29 specifies the path of control for the eight conditions.

<u>Condition 1:</u> An output data set is processed under QSAM in the move mode, and all channel programs execute normally.

The executor operates as follows:

- It issues a TRUNC macro-instruction to pass control to the PUT routine. (The PUT routine passes control to the end-of-block routine, which causes the channel program for the present buffer to be scheduled for execution. Control returns to the PUT routine, which returns control to this executor.)
- It awaits execution of the channel program for the present buffer.
- It tests the execution of the channel program and finds that it executed normally.
- It passes control to the end-of-volume routine of I/O support. (That routine passes control to the EOV/new volume executor, which returns control to the processing program.)

<u>Condition 2:</u> An output data set is processed under QSAM in the move mode, and a permanent error condition is encountered in the execution of a channel program.

The executor operates as follows:

- It issues a TRUNC macro-instruction to pass control to the PUT routine. (The PUT routine passes control to the end-of-block routine, which causes the channel program for the present buffer to be scheduled for execution. Control returns to the PUT routine, which returns control to this executor.)

- It awaits execution of the channel program, and finds that it encountered an error condition in its execution. It passes control to the synchronizing routine. (That routine finds the same error condition and passes control to the SYNAD/EOV executor (IGC0005E) by means of an SVC 55 instruction. The SYNAD/EOV executor finds that the error condition is a permanent error condition and returns control to the synchronizing routine, which returns control to the FEOV executor.)

- It issues an ABEND macro-instruction.

Condition 3: An output data set is processed under QSAM in the move mode, and an end-of-volume condition is encountered in the execution of a channel program.

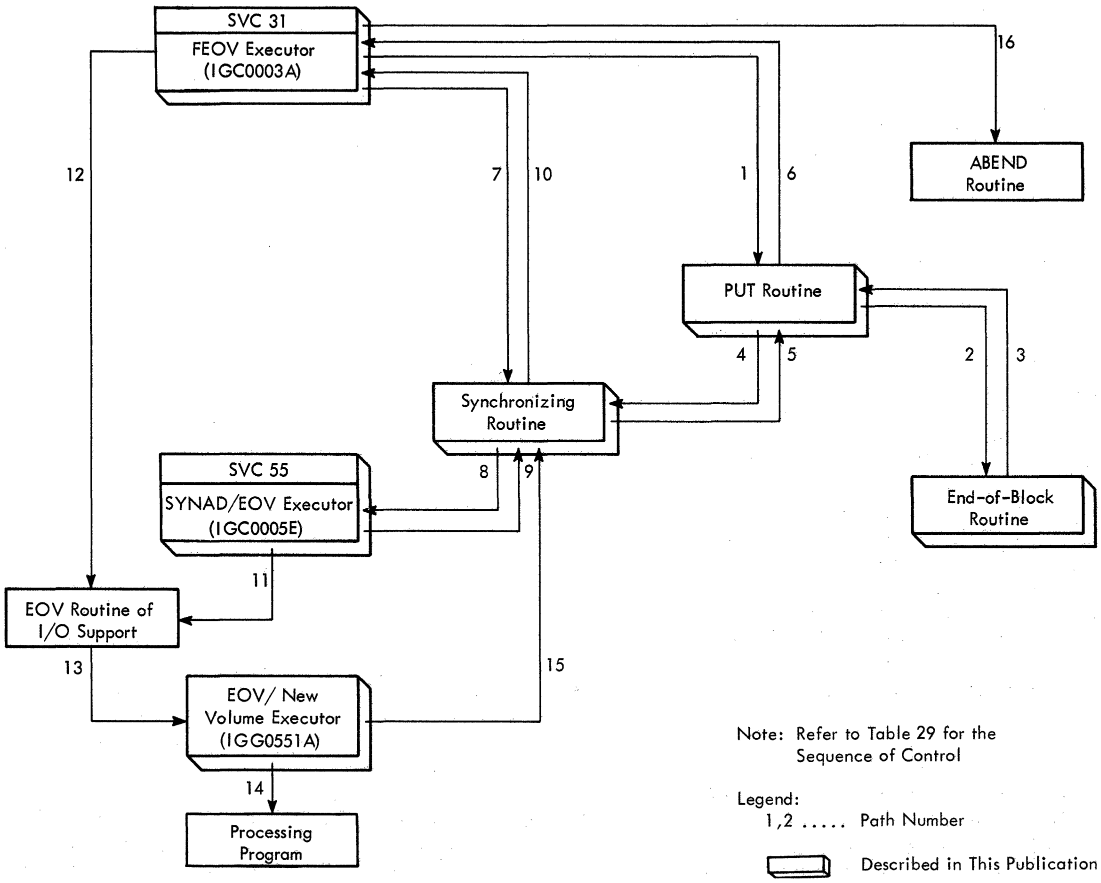
The executor operates as follows:

- It issues a TRUNC macro-instruction to pass control to the PUT routine. (The PUT routine passes control to the end-of-block routine, which causes the channel program for the present buffer to be scheduled for execution. Control returns to the PUT routine, which returns control to the FEOV executor.)

- It awaits execution of the channel program, and finds that it encountered an error condition in its execution.

- It passes control to the synchronizing routine. (The routine finds the same error condition and passes control to the SYNAD/EOV executor (IGC0005E) by means of an SVC 55 instruction. The SYNAD/EOV executor finds that the error condition is an end-of-volume condition and passes control to the EOV routine of I/O support. That routine passes control to the EOV/new volume executor, which returns control to the synchronizing routine. The synchronizing routine now returns control to the FEOV executor.)

- It passes control to the end-of-volume routine of I/O support. (That routine passes control to the EOV/new volume executor again, which now returns control to the processing program.)



Note: Refer to Table 29 for the Sequence of Control

Legend:
1,2 ..... Path Number
▭ Described in This Publication

Figure 18. Flow of Control Between the FEOV Executor and Other Control Program Routines

Table 29. Path and Sequence of Control of the FEOV Executor and Other Control Program Routines

Condition (a)	Sequence of Control (b)
1	1,2,3,6,12,13,14
2	1,2,3,6,7,8,9,10,16
3	1,2,3,6,7,8,11,13,15,10,12,13,14
4	1,2,3,4,5,6,12,13,14
5	1,2,3,4,8,9,10,16
6	1,2,3,4,5,6,7,8,9,10,16
7	1,2,3,4,8,11,13,15,10,12,13,14
8	1,2,3,4,5,6,7,8,11,13,15,10,12,13,14

Legend:
(a) - Refer to Appendix E for a description of the conditions.
(b) - Refer to Figure 18 for an identification of the routine passing control and the routine receiving control.

Condition 4: An output data set is processed under QSAM in the locate mode, and all channel programs execute normally.

The executor operates as follows:

- It issues a TRUNC and a PUT macro-instruction to pass control to the PUT routine. (The PUT routine passes control to the end-of-block routine, which causes the channel program for the present buffer to be scheduled for execution. The PUT routine then passes control to the synchronizing routine to obtain the next buffer. That routine finds that the channel program for the next buffer executed normally, and returns control to the PUT routine. The PUT routine returns control to the FEOV executor.)

- It awaits execution of the last channel program, and finds that the channel program executed normally.

- It passes control to the EOV routine of I/O support. (That routine passes control to the EOV/new volume executor, which returns control to the processing program.)

Condition 5: An output data set is processed under QSAM in the locate mode, and the execution of the channel program for the next buffer in the chain encountered a permanent error.

The FEOV executor operates as follows:

- It issues a TRUNC and a PUT macro-instruction to pass control to the PUT routine. (The PUT routine passes control to the end-of-block routine, which causes the channel program for the present buffer to be scheduled for execution. The PUT routine then passes control to the synchronizing routine to obtain the next buffer. The synchronizing routine finds that the channel program executed with an error condition and passes control to the SYNAD/EOV executor (IGC0005E), by means of an SVC 55 instruction. The SYNAD/EOV executor finds that the error condition is a permanent error condition, and returns control to the synchronizing routine. The synchronizing routine now returns control to the FEOV executor.)

- It issues an ABEND macro-instruction.

Condition 6: An output data set is processed under QSAM in the locate mode, and the execution of the channel program for any buffer other than the buffer specified in condition 5 encounters a permanent error.

The executor operates as follows:

- It issues a TRUNC and a PUT macro-instruction to pass control to the PUT routine. (The PUT routine passes control to the end-of-block routine, which causes the channel program for the present buffer to be scheduled for execution. The PUT routine then passes control to the synchronizing routine, which returns control to the PUT routine. The PUT routine returns control to the executor.)

- It awaits execution of the channel program for the last buffer and finds that the channel program executed with an error condition.

- It passes control to the synchronizing routine. (The routine finds the same error condition and passes control to the SYNAD/EOV executor (IGC0005E), by means of an SVC 55 macro-instruction. The SYNAD/EOV executor finds that the error condition is a permanent error condition and returns control to the synchronizing routine, which returns control to the FEOV executor.)

- It issues an ABEND macro-instruction.

Condition 7: An output data set is processed under QSAM in the locate mode, and the execution of the channel program for the next buffer in the chain encountered an end-of-volume condition.

The executor operates as follows:

- It issues a TRUNC and a PUT macro-instruction to pass control to the PUT routine. (The PUT routine passes control to the end-of-block routine, which causes the channel program for the present buffer to be scheduled for execution. The PUT routine then passes control to the synchronizing routine to obtain the next buffer. The synchronizing routine finds that the channel program executed with an error condition, and passes control to the SYNAD/EOV executor (IGC0005E), by means of an SVC 55 instruction. The SYNAD/EOV executor finds that the error condition is an EOV condition, and passes control to the EOV routine of I/O support. That routine passes control to the EOV/new volume executor, which passes control to the synchronizing routine. The synchronizing routine returns control to the PUT routine, which now returns control to the FEOV executor.)

- It passes control and the work area to the EOV routine of I/O support. (That routine passes control to the EOV/new volume executor again, which now returns control to the processing program.)

Condition 8: An output data set is processed under QSAM in the locate mode, and the channel program for any buffer other than the one specified in condition 7 encounters an end-of-volume condition.

The executor operates as follows:

- It passes control to the PUT routine. (The PUT routine passes control to the end-of-block routine, which causes the channel program for the present buffer to be scheduled for execution. The PUT routine then passes control to the synchronizing routine which returns control to the PUT routine. The PUT routine returns control to the FEOV executor.)

- It awaits execution of the channel program for the present buffer, and then finds that the channel program executed with an error condition.

- It passes control to the synchronizing routine. (The routine finds the same error condition and passes control to the SYNAD/EOV executor (IGC0005E) by means of an SVC 55 instruction. The SYNAD/EOV executor finds that the error condition is an EOV condition and passes control to the EOV routine of I/O support. That routine passes control to the EOV/new volume executor, which passes control to the synchronizing routine, which returns control to the FEOV executor.)

- It passes control, and the work area, to the EOV routine of I/O support. (That routine passes control to the EOV/new volume executor again, which now returns control to the processing program.)

Note: An EOV condition is found during the implementation of an FEOV macro-instruction in conditions 3, 7, and 8. The subsequent processing results in three volumes: Two volumes containing all the blocks scheduled for output by the FEOV macro-instruction and prior PUT macro-instructions, and a third volume available for writing new blocks.

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

READER'S COMMENTS

Title: IBM System/360 Operating System
Sequential Access Methods
Program Logic Manual

Form: Y28-6604-1

Is the material:	Yes	No
Easy to Read?	___	___
Well organized?	___	___
Complete?	___	___
Well illustrated?	___	___
Accurate?	___	___
Suitable for its intended audience?	___	___

How did you use this publication?

___ As an introduction to the subject

Other ___________________________

___ For additional knowledge

fold

Please check the items that describe your position:

___ Customer personnel
___ IBM personnel
___ Manager
___ Systems Analyst
___ Operator
___ Programmer
___ Customer Engineer
___ Instructor
___ Sales Representative
___ Systems Engineer
___ Trainee
Other______________

Please check specific criticism(s), give page number(s),and explain below:

___ Clarification on page(s)
___ Addition on page(s)
___ Deletion on page(s)
___ Error on page(s)

Explanation:

fold

FOLD ON TWO LINES,STAPLE AND MAIL
No Postage Necessary if Mailed in U.S.A.

staple st

fold f

FIRST CLASS
PERMIT NO. 81

POUGHKEEPSIE, N.Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
P.O. BOX 390
POUGHKEEPSIE, N. Y. 12602

ATTN: PROGRAMMING SYSTEMS PUBLICATIONS
DEPARTMENT D58

---

fold f

Printed in U.S.A. Y28-6604-1

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

sta

102664835