IBM

MVS/Extended Architecture
SAM Logic

Program Product

DFP
AMODE
DFP 31-bit
MVS/XA RMC
31-bit

Order No. LY26-3967-0
File No. S370-30

Data Facility Product 5665-XA2
Version 2
Release 1.0

MVS/Extended Architecture SAM Logic

Data Facility Product 5665-XA2
Version 2 Release 1.0

LY26-3967-0

First Edition (April 1985)

This edition applies to Version 2 Release 1.0 of MVS/Extended Architecture Data Facility Product, Program Product 5665-XA2, and to any subsequent releases until otherwise indicated in new editions or technical newsletters.

The changes for Version 2 support are summarized under "Summary of Amendments" following the preface. Specific changes are indicated by a vertical bar to the left of the change. These bars will be deleted at any subsequent republication of the page affected. Editorial changes that have no technical significance are not noted.

Changes are made periodically to this publication; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/370 and 4300 Processors Bibliography, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

Publications are not stocked at the address given below; requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California, U.S.A. 95150. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

PREFACE

This manual is intended for programming-support customer engineers and programmers who require specific information about queued sequential access method (QSAM), basic sequential access method (BSAM), and basic partitioned access method (BPAM) routines.

ORGANIZATION

This manual has the following parts:

"Introduction" describes the sequential access method (SAM) routines and includes a reference to Diagram A, "Sequential Access Method—Overview." This diagram lists the macro statements used with SAM programming techniques and directs the reader to appropriate diagrams and figures in other parts of the manual.

In "Method of Operation," SAM routines are described in the following categories:

- Queued sequential access method (QSAM) routines that cause storage and retrieval of data records arranged in sequential order.
- Basic sequential access method (BSAM) routines that cause storage and retrieval of data blocks arranged in sequential order.
- Basic partitioned access method (BPAM) routines that cause storage and retrieval of data blocks in a member of a partitioned data set. They can also 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 virtual storage.
- Problem determination that helps the user determine the causes of abends by providing more information on the reason for the termination.
- SVC routines that provide supervisor state operation for functions that cannot be done in the problem state or in the user's key.
- Task recovery routines that provide explicit validity checking for SVC routines that experience program checks or other abend conditions.

"Program Organization and Flow of Control" contains diagrams that describe the organization and flow of control of the SAM routines.

"Directory" lists the names of the sequential access method modules in alphabetic order. Each entry contains the module name, type, CSECT name, SVC entry (if any), and references to figures and appendixes in other parts of the manual that have information about the module.

"Data Areas" shows how various control blocks are used in QSAM and BSAM. This section also describes the access method save area for user totaling and the job entry subsystem (JES) compatibility interface control block. "Data Areas" does not describe in detail all fields of the system control blocks referred to in this manual. For more detailed information about

system control blocks, see MVS/Extended Architecture Data Areas-JES2, LYB8-1191, and MVS/Extended Architecture Data Areas-JES3, LYB8-1195.

"Diagnostic Aids" contains diagrams of control blocks and an abend codes cross-reference table.

"Appendixes" describe channel programs for direct-access storage, and BDAM create channel programs.

PREREQUISITE KNOWLEDGE

To use this book efficiently, you should be familiar with the following topics:

- Basic concepts of data management
- Processing sequential and partitioned data sets

PREREQUISITE READING

The above topics are discussed in MVS/Extended Architecture Data Administration Guide, GC26-4140.

RELATED PUBLICATIONS

Within the text, references are made to the publications listed in the table below:

Short Title	Publication Title	Order Number
ACF/TCAM Diagnosis Guide	Advanced Communications Function for TCAM, Version 2 Diagnosis Guide	SC30-3137
ACF/TCAM Diagnosis Reference	Advanced Communications Function for TCAM, Version 2 Diagnosis Reference	LY30-3052
Data Administration: Macro Instruction Reference	MVS/Extended Architecture Data Administration: Macro Instruction Reference	GC26-4141
Data Areas	MVS/Extended Architecture Data Areas (MVS/JES2)	LYB8-1191
Data Areas	MVS/Extended Architecture Data Areas (MVS/JES3)	LYB8-1195
Debugging Handbook	MVS/Extended Architecture Debugging Handbook, Volumes 1 through 5	LC28-1164[1] LC28-1165 LC28-1166 LC28-1167 LC28-1168
IBM 3800 Printing Subsystem Programmer's Guide	IBM 3800 Printing Subsystem Models 3 and 8 Programmer's Guide	SH35-0061

Note:

[1] All five volumes may be ordered under one order number, LBOF-1015.

Short Title	Publication Title	Order Number
Initialization and Tuning Guide	MVS/Extended Architecture System Programming Library: Initialization and Tuning	GC28-1149
JES2 Logic	MVS/Extended Architecture JES2 Logic	LY24-6008
Open/Close/EOV Logic	MVS/Extended Architecture Open/Close/EOV Logic	LY26-3966
OS/VS Logic for IBM 3890 Document Processor	OS/VS Logic for IBM 3890 Document Processor	SY24-5163
Service Aids	MVS/Extended Architecture System Programming Library: Service Aids	GC28-1159
SYS1.LOGREC Error Recording	MVS/Extended Architecture System Programming Library: SYS1.LOGREC Error Recording	GC28-1162
System Codes	MVS/Extended Architecture Message Library: System Codes	GC28-1157
System-Data Administration	MVS/Extended Architecture System-Data Administration	GC26-4149
System Logic Library	MVS/Extended Architecture System Logic Library, Volumes 1 through 16	SY28-1208 through LY28-1266
System Messages	MVS/Extended Architecture Message Library: System Messages, Volumes 1 and 2	GC28-1376 and GC28-1377
VIO Logic	MVS/Extended Architecture VIO Logic	LY26-3900

SUMMARY OF AMENDMENTS

RELEASE 1.0, APRIL 1985

NEW DEVICE SUPPORT

- Module descriptions for IGX00030, IGX00031, and IGX00032 have been added.
- Updates to support the IBM 4248 and 3263 Model 5 Printers have been made to the Printer Device Characteristics Table (IGGPDC) and the SETPRT Parameter List (IHASPP) under "Data Areas" on page 210, and the SETPRT Executor Return/Reason Codes and Messages Table under "Diagnostic Aids" on page 242.

NEW PROGRAM SUPPORT

A description of the subsystem CICB has been added to "Data Areas" on page 210.

VERSION 2 PUBLICATIONS

The Preface includes new order numbers for Version 2.

CONTENTS

FIGURES

INTRODUCTION

Sequential access methods (SAM) are programming techniques for transferring data arranged in sequential order between virtual storage and an input/output device. This manual describes five groups of sequential access method routines. They are:

- 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 works 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. See Diagram F for information about the flow of control for QSAM routines.

A processing program using BSAM routines works 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. See Diagram G for flow of control information about BSAM routines.

A processing program that uses BSAM or QSAM to access SYSIN or SYSOUT data sets invokes a special subset of SAM routines called SAM-SI (SAM Subsystem Interface). These routines operate as a compatibility interface to job entry subsystems, such as JES2, that control these data sets. See Diagram M in "Program Organization and Flow of Control" for information about the flow of control in SAM-SI routines for BSAM and QSAM.

A processing program using BPAM routines also works with blocks of data. For output, BPAM routines construct and cause writing of entries in the directory; for input, BPAM routines search for and read entries from the directory. To read and write the blocks of the members, a processing program uses the BSAM routines. Flow of control for the BPAM routines is shown in Diagram G.

Sequential access method executors are modules that operate with the OPEN and CLOSE routines. When a data control block is opened, an executor constructs control blocks and loads the access method routines. The access method routines reside in the link pack area.

When the end of a data set or volume is reached, an EOV SVC is issued to process the pending input/output blocks. The executors described are:

- OPEN executors
- CLOSE executors

Buffer-pool management routines form buffers in virtual storage and return virtual 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 together form a pool of buffers in virtual storage. However, the GETPOOL routine also obtains the virtual storage space for the buffer pool. Virtual 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 virtual-storage space used for a buffer pool.

Diagram A in "Program Organization and Flow of Control" lists the macro statements that are used with sequential access method programming techniques. The diagram also refers to figures in other portions of the manual that describe the SAM routines, appendages, and executors associated with each macro statement.

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 seven types of QSAM routines:

- GET routines
- PUT routines
- End-of-block routines
- Synchronizing and error-processing routines (including the IBM 3211 and 3203 printer retry asynchronous-error-processing routines)
- Appendage routines
- Control routines
- SVC Routines

Diagram F, "QSAM Flow of Control," shows the relationship of QSAM routines to other portions of the operating system and to the processing program.

GET ROUTINES

GET routines determine the address of the next input record by referring to the DCB. In update mode, the next output record is the last input record.

If the American National Standard Code for Information Interchange (ASCII) is used, the GET routine (if it is specified in the DCB) will accept a record with a block prefix. The GET routines do not present the block prefix to the processing program; the block prefix is specified by the BUFOFF option in the DCB. For more information on block prefix and record formats for ASCII, see Data Administration Guide.

Because there is an unused byte at the beginning of each segment descriptor word (SDW), the GET routines that process records in the ISO/ANSI/FIPS spanned record format must make record address adjustments. The unused byte results from the conversion of the 5-byte ISO/ANSI/FIPS segment control word (SCW) to the 4-byte IBM SDW.

The GET routine descriptions that follow are accordingly grouped as:

- Simple-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 DCBRECAD field (address of the next record); the ending address is in the DCBEOBAD field (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 for tape and unit record devices, and by IGG0193B for direct-access storage devices. 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 and DCBEOBAD fields are equal. For ISO/ANSI/FIPS fixed block format, an EOB condition exists when the next logical record of a block consists of all X'5F's. In the move operating mode, the buffer can be scheduled for refilling as soon as the last record is moved out; thus, an EOB test is made after moving each record, so that the buffer can be scheduled for refilling 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; thus, an EOB test is made on entry into the routine to determine 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 wanted, the processing program issues a RELSE macro instruction. Control passes to a RELSE routine that sets an EOB condition. When records are spanned, one or more blocks can be skipped to find the first record in a new block.

If QSAM is used with a DCB opened for input, update, or readback, the OPEN executor primes (that is, schedules for filling) the buffers. 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 usual way.

Upon return from the synchronizing and error-processing routine, the GET routines, which may be loaded for tape data sets, test to determine if the buffer contains a DOS checkpoint record. If a DOS checkpoint record is indicated, ECB posted X'50', the GET routine branches to the end-of-block routine to reschedule the buffer for refilling and then branches back to the synchronizing routine to test the next buffer.

Figure 1 on page 6 lists the simple-buffering GET routines 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. Figure 1 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.

Access Method Options	Selec-tions	Selec-tions	Selec-tions	Selec-tions	Selec-tions	Selec-tions
INPUT, GET	X X X	X X X	X X		X X	X X X
RDBACK, GET			X	X X X		
Locate mode	X X X		X	X	X X	
Move mode		X X X	X X	X X		X
Data mode						X
Fixed-length record format	X	X	X X	X		
Undefined-length record format	X	X	X	X X		
Variable-length or record format-D	X	X			X X	X X
Spanned records					X X	X X
* or DATA on DD statement						X
Card reader, only a single, buffer CNTRL			X X			
Logical record interface					X	
GET Modules						
IGG019AA	AA AA					
IGG019AB	AB					
IGG019AC		AC AC				
IGG019AD		AD				
IGG019AG			AG AG			
IGG019AM			AM	AM		
IGG019AN				AN AN		
IGG019BO					BO	
IGG019DJ						DJ
IGG019FB					FB	
IGG019FD						FD
IGG019FF						FF

Figure 1. Module Selection—Simple-Buffering GET Modules

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 if a new buffer is needed. When the OPEN executor primes the buffers, it schedules all buffers except one and sets an EOB condition. For ISO/ANSI/FIPS, an EOB condition exists when the next logical record in a block consists of all padding characters (X'5F's). The first logical record in a block must not consist of all padding characters.
- If no EOB condition exists, the GET routine 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, the GET routine 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.

GET Module IGG019AB: Module IGG019AB presents the processing program with the address of the next variable-length or format-D 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 or record format-D (unblocked or blocked), unspanned

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 if a new buffer is needed. When the OPEN executor primes the buffers, it schedules all buffers except one and sets an EOB condition. For ISO/ANSI/FIPS, an EOB condition exists when the next logical record in a block consists of all padding characters (X'5F's). The first logical record in a block must not consist of padding characters.
- 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

The DCB does not, however, specify 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 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 or format-D 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 or record format-D (unblocked or blocked), unspanned

The DCB does not, however, specify 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, issues a BALR to the input-synchronizing and error-processing routine, module IGG019AQ, 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 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. For ISO/ANSI/FIPS, logical records consisting of all padding characters (X'5F's) are skipped. That is, when these records are encountered in a block, they are treated as padding and processing begins with the next logical record in backward sequence.

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 2 on page 13 illustrates the ordering of records using this module. When reading backward 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.
- For ISO/ANSI/FIPS, logical records consisting of all padding characters (X'5F's) are skipped. That is, when these records are encountered in a block they are treated as padding, and processing begins with the next logical record in backward sequence.

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 2 illustrates the ordering of records using modules IGG019AM and IGG019AN.

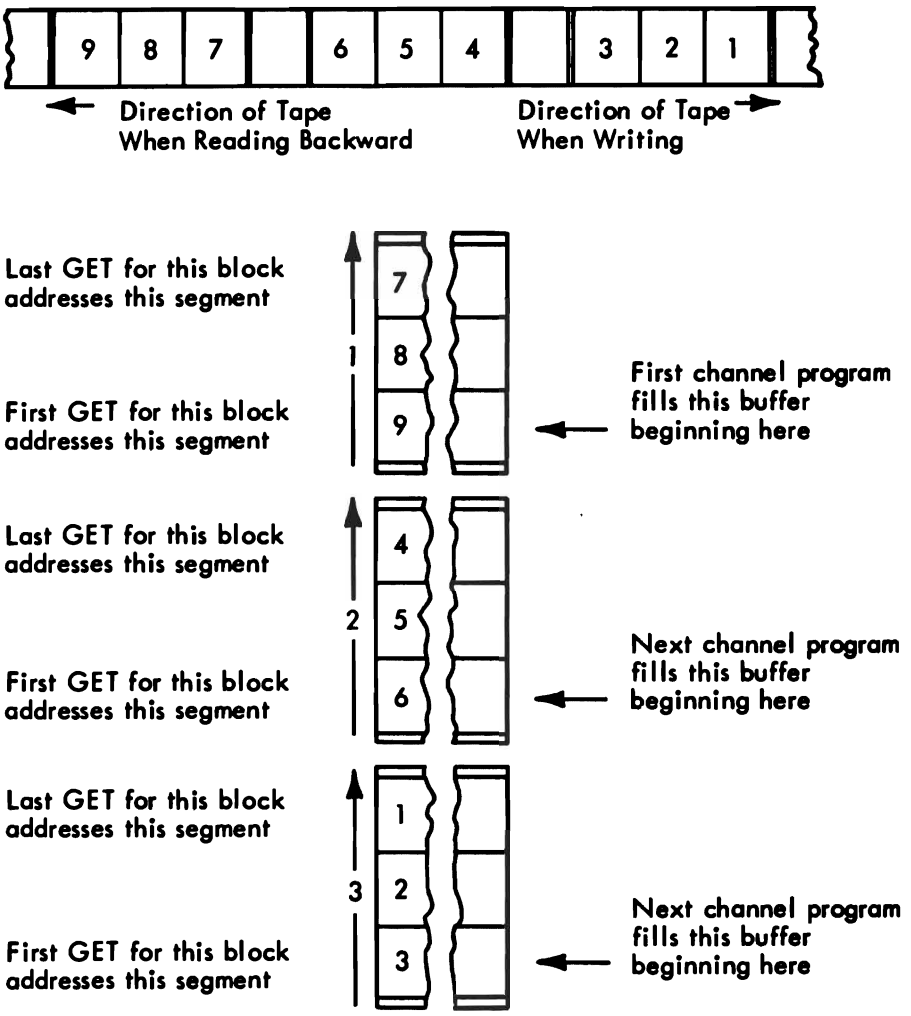


Figure 2. Order of Records Using GET Routines for Data Sets Opened for RDBACK (IGG019AM, IGG019AN)

GET Module IGG019BO: Module IGG019BO 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 spanned (unblocked or blocked) record format

Logical record interface

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 the 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 if new buffer is needed. When the OPEN executor primes the buffers, it schedules all buffers except one and sets an EOB condition. If ISO/ANSI/FIPS spanned records are being processed, the record address is adjusted in order to ignore the unused byte that results from the conversion of the 5-byte ISO/ANSI/FIPS segment control word (SCW) to the 4-byte IBM segment descriptor word (SDW).
- If no EOB condition exists, it tests whether the next record segment contains a complete record.
- If it is a complete record, the routine presents the address of the next record to the processing program and returns control to the processing program. If the extended version of logical record interface (XLRI) is being used, the three low-order bytes of the first four bytes of a logical record are used to indicate the length of the record including the first four bytes.
- If it is the first segment of a spanned record, the routine moves the segment to the record area with the proper alignment, sets the EOB condition, and determines the address of the next record and whether a buffer is ready for refilling.
- If it is a segment that follows another segment of a spanned record, the routine moves the segment (without the segment descriptor word) next to the previous segment in the record area, and updates the count in the record area. This step continues until the entire logical record has been assembled in the record area. If an EOB condition occurs during this process, the routine determines the address of the next record and whether a buffer is ready for refilling. When the entire logical record is assembled, the routine sets the spanned record flag in the IOB, presents the address of the assembled record, and returns control to the processing program.
- If an EOB condition exists, it issues a BALR instruction to pass the present buffer to the EOB 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 IGG019A0). The routine then obtains and interrogates the first record segment of the new buffer. If it is a complete record, the routine presents the address of the next record to the processing program 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 EOB condition.
- It sets a release bit in the DCBRECAD of the DCB.
- It returns control to the processing program.

The RELSE routine sets a release 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. After obtaining the new buffer as a result of RELSE, the GET routine interrogates the SDW of the first segment to determine if it is the first segment of a record (bit 6 in third byte of SDW must

be 0); if not, the routine skips to the next SDW and checks it. This continues until an acceptable segment is found. The routine then processes the GET request in the usual way. The procedure may result in one or more additional blocks being passed.

GET Module IGG019DJ (SYSIN/SYSOUT): Module IGG019DJ interfaces with a job entry subsystem to provide the next record from the system input stream to the processing program.

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

Input (* or DATA specified on the DD statement)

and the DCB specifies:

GET

Simple buffering

Locate or move operating mode

Fixed, undefined, or variable-length record format

The module consists of a GET routine and a RELSE routine. See Diagram M for an overview of the SAM-SI processing for QSAM.

This module also contains a PUT routine as described in "Simple Buffering PUT Routines" (see Figure 4 on page 26). The GET routine operates as follows:

- It receives control when a GET macro instruction is encountered in the processing program.
- It determines the type of get request and initializes the input area address in the request parameter list (RPL). For move mode, RPLAREA contains the address of the processing program work area (the contents of register 0 on entry); for locate mode, RPLAREA contains the address of a buffer from the DCB buffer pool.
- If the GET request is for variable-length records, RPLAREA is adjusted to allow space for a record descriptor word (RDW) in the first four bytes of the work area.
- It passes control to the job entry subsystem (JES) for data transfer by issuing a GET macro instruction against the RPL. The return code in register 15 is tested upon return from the JES.
- For an exceptional condition, RPLRTNCD and RPLERRCD are examined to determine the type of failure.
- If end-of-data is detected, the appropriate registers are loaded and saved, then an unconditional branch is taken to the synchronizing module, IGG019AQ (see Figure 11 on page 60), for EODAD and concatenation processing.
- If an error condition is detected, control is passed to the error-processing module, IGG019AH (see Figure 12 on page 67). If control is returned to this routine and DCB EROPT is SKIP, the GET request is reissued. Otherwise, control is returned to the processing program.
- For normal completion, it places the record address from the RPLAREA field into register 1. If the SAM request was for a variable-length record, the record descriptor word field is created, by using the value returned in the RPLRLEN field. Registers are restored and control is returned to the processing program.

The RELSE routine receives control when a RELSE macro instruction is issued. Module IGG019DJ does no processing for this macro instruction. Control is returned to the processing program by IGG019DJ.

GET Module IGG019FB: Module IGG019FB 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 format (unblocked or blocked) record, spanned

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

The GET routine operates as follows:

- It receives control when the processing program issues a GET macro instruction.
- It determines the address of the next record segment and tests for an EOB condition to determine whether a buffer is ready for refilling and also whether a new buffer is needed. When the OPEN executor primes the buffers, the executor schedules all buffers except one and sets an EOB condition.
- If no EOB condition exists, the routine presents the address of the next record segment to the processing program.
- If an EOB condition exists or if a DOS-type null segment (where the high-order bit of the record descriptor word is on) is encountered, the routine issues a BALR instruction to pass the current buffer to the EOB routine. The EOB routine schedules the buffer 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 determines if the EOB routine was entered because of a RELSE macro instruction. If so, the GET routine checks the first record segment to determine if it is a member of a previous logical record. If it is, the GET routine continues to look for a record segment that is not a member of a previous record. Such a segment is considered the first record of the new buffer. (Note, however, that this could cause reentry into the EOB routine and result in one or more entire blocks being skipped.) The GET routine then presents the address of the first record segment 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 sets the high-order 4 bits of DCBRECAD to 1's and returns control to the processing program.

GET Module IGG019FD: Module IGG019FD moves the next variable-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, spanned

The DCB does not, however, specify the CNTRL macro instruction.

The module consists of a GET and a RELSE routine.

The GET routine operates as follows:

- It receives control when the processing program issues a GET macro instruction.
- It tests for an EOB condition to determine whether a new full buffer is needed. When the OPEN executor primes the buffers, the executor also sets an EOB 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 segment to the user's work area.
- If no EOB condition exists, the routine moves the first record segment to the user's work area.
- If a DOS-type null segment (where the high-order bit of the record descriptor word is on) is encountered, that buffer is rescheduled by passing control to the EOB routine. Processing continues as if an EOB condition exists as described above.
- If more record segments are required, the routine moves them, without the segment descriptor words, to the part of the user's work area that is contiguous with the previous record segment. The routine also updates the DCBLRECL field and the logical-record-length field in the record descriptor word (RDW) in the user's work area. These fields then reflect the total logical-record length after additional record segments have been moved. This procedure continues until the routine has moved the entire logical record. An EOB condition can occur during this procedure.
- When ISO/ANSI/FIPS spanned records are being processed, the address of the starting byte must be adjusted in order to ignore the unused byte resulting from the conversion of the 5-byte ISO/ANSI/FIPS segment control word (SCW) to the 4-byte IBM segment descriptor word (SDW).
- The routine tests for a new EOB condition to determine whether a buffer is empty and ready for refilling. For unblocked records, the EOB condition exists after every entry to the GET 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 EOB routine. The EOB routine then schedules the buffer for refilling and returns control to the processing program.

The RELSE routine sets the high-order 4 bits in the DCBRECAD field to 1's so that the GET routine passes the buffer for refilling and so that the next time the GET routine is entered, it obtains a new full buffer. After obtaining the new buffer, the GET routine interrogates the segment descriptor word (SDW) of the first record segment. The routine thus determines if the segment is the first segment of a record. If it is, bit 6 of the third byte of the SDW will be 0. If not, the GET routine skips to the next SDW and checks it. This procedure continues until an acceptable segment is found. Then the GET routine processes the GET macro instruction in the usual way. The procedure can result in one or more additional blocks being passed.

GET Module IGG019FF: Module IGG019FF moves the data portion of the next variable-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

Data operating mode

Variable-length (unblocked or blocked) record format, spanned

The DCB does not, however, specify the CNTRL macro instruction.

The module consists of GET and RELSE routines.

The GET routine operates as follows:

- It receives control when the processing program issues a GET macro instruction.
- It tests for an EOB condition to determine whether a new full buffer is needed. When the OPEN executor primes the buffers, the executor also sets an EOB 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 data portion of the first record segment to the work area.
- If no EOB condition exists, the routine moves the data portion of the first record segment to the user's work area.
- If more segments are required, the routine moves them, without the segment descriptor word, to the part of the user's work area that is contiguous with the previous record segment. The routine also updates the DCBLRECL field to reflect the current total logical record length. This procedure continues until the routine has moved the entire logical record. An EOB condition can occur during this procedure.
- When ISO/ANSI/FIPS spanned records are being processed, the address of the starting byte must be adjusted one position in order to ignore the unused byte resulting from the conversion of the 5-byte ISO/ANSI/FIPS segment control word (SCW) to the 4-byte IBM segment descriptor word (SDW).

- The routine tests for a new EOB condition to determine whether a buffer is 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 EOB routine. The EOB routine then schedules the buffer for refilling and returns control to the processing program.

The RELSE routine sets the high-order 4 bits in the DCBRECAD field to 1s so that the GET routine passes the buffer for refilling and so that the next time the GET routine is entered, it obtains a new full buffer. After obtaining the new buffer, the GET routine interrogates the segment descriptor word (SDW) of the first record segment. The routine thus determines if the segment is the first segment of a record. If it is, bit 6 of the third byte of the SDW will be 0. If not, the GET routine skips to the next SDW and checks it. This procedure continues until an acceptable segment is found. Then the GET routine processes the GET macro instruction in the usual manner. The procedure can result in one or more additional blocks being passed.

Parallel Input Processing Routine

The QSAM parallel input processing routine provides to the user an input record from a queue of equal priority, sequential data sets. The routine supports input processing; simple buffering; locate or move mode; and fixed-length, variable-length, or undefined-length records. Track overflow and spanned records are not supported.

Parallel Input Processing Module IGG019JD: Module IGG019JD uses the parallel data address block (PDAB) to maintain a list of data control blocks, addresses, and a corresponding wait parameter list of ECB addresses. DCB addresses are added to the PDAB by the OPEN routines and are removed by the CLOSE routines. A count of the maximum number of DCB entries allowable is assembled in the PDAB.

The address of the DCB entry from which the previous record was provided is obtained from the PDAB, and each succeeding DCB entry is processed until an available logical record is found, or until each data set is found to have reached an EOB condition, and the next block of data is not available.

An EOB condition is detected when DCBEOBAD is greater than or equal to DCBREGAD for the move mode, when DCBEOBAD is greater than or equal to DCBECAD plus DCBLRECL for the locate mode, or when the first 4 bits of the DCBIOBA are set to ones for the RELSE function.

The next block is not available when the ECB for the next IOB is not posted as complete. The location of the next IOB is obtained from the current IOB - 8, and the location of its corresponding ECB is obtained from IOB + 4.

When the ECB is not posted as complete, its address is stored in the wait parameter list in the PDAB. When no record is available from the queue of data sets, a WAIT is issued for the list of ECB addresses in the PDAB. When control is returned, the completed event is located from the list of ECB addresses.

When a record is available, the DCB address and the user's data area address are passed to the DCB get routine.

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 addresses of the buffer.

If a PUTX macro instruction addresses 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 addresses 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 are 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) and the DCBEOBAD (the address of the end of the block) fields 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 IOBNFLG1 flag to indicate whether an update (that is, write and refill) or a read (that is, a refill) is to take place. The "empty and refill" operation writes out of the buffer and reads into that same buffer. When the end-of-block routine schedules the IOB for the buffer to be processed by the SIO/pagefix appendage, that appendage inspects the IOB flags. The SIO/pagefix appendage builds an appropriate channel program, based on the IOB flags: an update write of the buffer followed by a read into the same buffer, or a read into the buffer.

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; otherwise, the flag is off. To schedule the buffer for empty-and-refill, the GET routine leaves the update flag on. To schedule the buffer for refill only, the GET routine sets the read flag on. The end-of-block condition

that triggers this processing also causes control to pass to the end-of-block routine, module IGG019TV, for issuing the EXCPVR 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. 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 the update mode GET routine gains control, it processes this condition in its normal manner.

Figure 3 on page 21 shows the update mode GET routines and the access conditions that must be specified in the DCB to select a particular routine. The OPEN executor loads the selected routine and places its address into the DCBGET field of the DCB.

Access Method Options	**Selections**						
Update, GET	X	X	X	X	X	X	X
Fixed-length record format	X	X					
Variable-length record format			X	X		X	X
Undefined-length record format					X		
Blocked record format	X		X			X	
Unblocked record format		X		X	X		X
Locate operating mode						X	X
Logical record interface						X	X
GET Modules							
IGG019AE[1]	AE	AE	AE	AE	AE		
IGG019BN						BN	BN

Figure 3. Module Selector—Update-Mode GET Modules

Note to Figure 3:

[1] This module also carries the Update-Mode PUTX routine

GET Update Module IGG019AE: Module IGG019AE presents the processing program with the next input record and flags the IOB if the block is to be updated. 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 if 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, format-D, 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 and if the buffer is to be emptied and refilled, and:
  - If entry is not from CLOSE or FEOV, the GET routine passes control to the end-of-block routine to cause scheduling of the buffer.
  - If entry is from CLOSE or FEOV, the GET routine sets the IOB to indicate "write-only." The GET 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, 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.

GET Update Module IGG019BN: Module IGG019BN presents the processing program with the next input record, flags the IOB if the block or a spanned record 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

Locate operating mode

Variable-length spanned (blocked or unblocked) record format

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

The GET routine operates as follows:

- It receives control when a GET macro instruction is encountered in a processing program.
- It tests whether EOV has occurred while processing a spanned record.
- If EOV has occurred and the record is not to be updated, it sets a bit in the DCBIOBAD field of the DCB to indicate that the old DEB, whose address was saved by the EOV routine, can be freed. It then issues an FEOV macro instruction to free the virtual storage assigned to this DEB.
- If EOV has occurred and the record is to be updated, it restores the address to read back the block that contains the beginning segment of the record. The current IOB is modified to function as if only one IOB exists. It then issues an FEOV macro instruction to cause the previous volume to be mounted and the data management count to be reset.
- On return of control from the FEOV routines, it operates as if no EOV has occurred.
- If EOV has not occurred, it continues on to the next step.
- It tests whether a spanned record is to be updated.
- If it is not to be updated, it obtains the length of the previous record segment from the DCBLRECL field in the DCB, or the SDW if it was a spanned record.
- It determines the address of the next record segment and tests for an EOB condition to determine whether the buffer can be scheduled and if 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 tests the next record segment for a complete record.
- If it is a complete record, the routine presents the address of the next record, determines the length of the record, places it in the DCBLRECL field, and returns control to the processing program.
- If it is the first segment of a spanned record, the routine saves the track address of the block that contains this segment, the position of the segment in the block, and the alignment of the segment in the record area. The routine obtains the track address of the block by copying the IOBSEEK associated with the next IOB, the position of the segment by subtracting the buffer address from the current record address, and the alignment of the segment by using the low-order byte of the current record address. The routine then moves the first segment to the record area and sets the EOB condition. It determines the address of the next record, whether a new buffer can be scheduled, and if a new buffer is needed.
- If it is a segment that follows another segment of a spanned record, the routine combines the segment (without the SDW) contiguous with the previous segment in the record area. The count in the record descriptor word (RDW) in the record area is updated to include the total count. This process continues until the entire logical record has been assembled. An EOB condition may occur during this process, in which case the routine determines the address of the next

record, whether a new buffer can be scheduled, and if a new buffer is needed. When the entire logical record has been assembled, the routine sets the spanned-record flag in the IOB, presents the address of the assembled record in the record area, places the length of the record (which is obtained from the RDW in the record area) in the DCBLRECL field, and returns control to the processing program.

- If an EOB condition exists, control is passed to the end-of-block routine to schedule a buffer.

- On return of control from the EOB routine, the routine passes control to the update-synchronizing and error-processing routine, module IGG019BQ, to obtain a new full buffer.

- On return of control from the synchronizing routine, the routine interrogates the next record segment and saves the track address of the block that contains the record, the position of the segment in the block, and the alignment of the segment in the record area. The routine then moves the first segment to the record area and sets the EOB condition.

- If a spanned record is to be updated, the routine restores the track address to read back the block that contains the beginning segment of the record. The current IOB is modified to function as if only one IOB exists.

The routine next tests to determine if any previous I/O operation has completed. If no previous I/O operation has completed, the routine issues WAIT against the ECB in the ICQE.

The routine next tests to determine if the "EXCPVR needed" flag is on and, if not, sets the "end of file" and "EXCPVR needed" flags on. The routine turns off the "spanned record" flag in the IOB, sets the IOB to READ-ONLY and SEGMENT, and passes control to the end-of-block routine.

- On return of control from the EOB routine, the routine passes control to the update-synchronizing and error-processing routine, module IGG019BQ, to obtain a new full buffer.

- On return of control from the synchronizing routine, the routine repositions the pointers to the beginning segment of the record and moves that portion of the record from the record area to the segment in the buffer. (A count is kept of the number of bytes of data moved.)

- If more segments are to be updated, the routine moves that portion of the record from the record area to the succeeding segments in the buffer. (The total count of the data moved is updated with each move.) This process continues until the entire logical record has been segmented. If an EOB condition occurs during this process, the routine tests whether a spanned record is to be updated. When the entire logical record has been segmented, the routine turns off the segment flag in the IOB, restores the link field in the IOB, obtains the address of the next record segment, and determines whether a new buffer can be scheduled and is needed.

When the entire logical record has been segmented (except for the last segment in the current buffer, which has not been updated), the routine turns off the "segment" flag in the IOB, restores the link field in the IOB, and tests the "end of data" flag to determine whether the "EXCPVR needed" flag was off when I/O was quiesced. If the "end of data" flag is on, the routine sets the IOB's "write flag" and passes control to the end-of-block routine to cause an update write without a refill read for the buffer containing the last segment. When the end-of-block routine returns, the GET routine sets the IOB flags to indicate "updating not required." The ICQE's "EXCPVR needed"

flag is also zeroed. If the "end of data" flag is off, the "EXCPVR needed" flag is set on.

The RELSE routine operates as follows:

- It receives control when a RELSE macro instruction is encountered in the processing program.
- It sets an EOB condition.
- It sets a release bit in the DCBRECAD field of the DCB.
- It returns control to the processing program.

The RELSE routine sets a release 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. After obtaining the new buffer as a result of RELSE, the GET routine interrogates the SDW of the first segment to determine if it is the first segment of a record (bit 6 in the third byte of the SDW must be 0); if not, the routine skips to the next SDW and checks it. This continues until an acceptable segment is found. The routine then processes the GET in the usual way. This procedure may result in one or more additional blocks being passed.

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.

Note: When a RELSE macro instruction is issued after a spanned record is written with a PUTX macro instruction, this routine branches to the GET routine to write the last record (the spanned record) and then releases the block that contains the last segment of that spanned record.

PUT ROUTINES

Some of the general characteristics of the PUT routines are described in Diagram B, "QSAM GET and PUT Routines." A specific PUT routine is selected for each data set on the basis of access method options specified by the processing program. The options examined are in the OPEN statement parameter list and the data set attributes described in the DCB.

The OPEN executors (see Diagram D, "SAM OPEN Executors") select and load the modules that are required for a particular data set.

The access method options that determine which PUT modules are selected when Simple buffering is used are described in Figure 4 on page 26. For update mode, the PUTX routine resides in the GET module for update mode. See Figure 3 on page 21 (under "Update Mode GET Routine") for information about the update mode PUTX routine.

For information about the flow of control through the QSAM routines, see Diagram F, "QSAM Flow of Control."

Access Method Options	Selections										
Output, PUT/PUT	X	X	X	X	X	X	X	X	X	X	X
Locate operating mode	X	X	X				X		X		
Move operating mode				X	X	X					
Data operating mode								X		X	
Fixed-length record format	X			X							
Undefined-length record format		X			X						
Variable-length or record format-D			X			X	X	X	X	X	
Spanned records							X	X	X	X	
Logical record interface							X				
SYSOUT specified on DD statement											X
PUT Modules											
IGG019AI	AI	AI									
IGG019AJ			AJ								
IGG019AK				AK	AK						
IGG019AL						AL					
IGG019BP							BP				
IGG019DJ											DJ
IGG019FG								FG			
IGG019FJ									FJ		
IGG019FL										FL	

Figure 4. Module Selector—Simple-Buffering Put Modules

Simple-Buffering PUT Routines

Simple-buffering PUT routines use buffers whose ending address and the address of the next or current record are pointed to by the DCB. The address of the next record is in the DCBRECAD field (address of the next record); the ending address is in the DCBEOBAD field (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 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 DCBBLKSI 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 the move operating mode is used, a PUT routine establishes that an EOB condition exists for the present buffer after the routine has moved the record into the buffer. If the locate operating mode is used, 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 the move operating mode is used, 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 the locate operating mode is used, 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, unspanned, an EOB condition occurs when the length of the next record exceeds the buffer balance; that is, when the record length exceeds the space remaining in the buffer. If the user has specified move mode for unspanned records, 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 the user has specified locate mode for unspanned records, the PUT routine establishes that an EOB condition exists when the value stated in the DCBLRECL field exceeds the buffer balance.

For variable-length blocked records, spanned, the next record is segmented. The first record segment is used to fill the buffer when 5 or more bytes remain in the buffer. When fewer than 5 bytes remain in the buffer, an EOB condition occurs.

For ISO/ANSI/FIPS variable-length spanned records, five bytes are used for the segment control word (SCW). An extra byte is

saved at the beginning of each segment. The succeeding four bytes are processed in the normal manner, but the end-of-block routine uses the extra byte when it converts the IBM 4-byte segment descriptor word (SDW) to the 5-byte ISO/ANSI/FIPS segment control word (SCW).

For variable-length spanned records using extended logical record interface (XLRI), the 3-byte length field is used to specifiy the exact length in bytes instead of the normal 2-byte length field. The DCB LRECL specifies the maximum logical record length in multiples of 1024.

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.

Figure 4 on page 26 lists the PUT routines and the conditions that cause a particular routine to be read. The OPEN executor selects one of the routines, loads it, and places its address into the DCBPUT fields.

PUT Module IGG019AI: Module IGG019AI presents the processing program with the address of the next available buffer segment for a fixed-length or an 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 if 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-error-processing routine, module IGG019AR, and determines the address of the

first segment of the new buffer. The PUT routine then presents this address and returns control to the processing program.

The TRUNC routine causes an EOB condition by setting the DCBRECAD and DCBEODAD 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 or format-D 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

Variable-length or record format D (unblocked or blocked), unspanned

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 if a new empty buffer is needed, by 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 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 exists, 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 moves the DCB's LRECL field from the input DCB to the output DCB.
- 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 or format-D 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 or record format-D (unblocked or blocked), unspanned

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 instead 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-synchronzing-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.

PUT Module IGG019BP: Module IGG019BP 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

Variable-length spanned (unblocked or blocked) record format

Logical record interface

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.
- If extended logical record interface (XLRI) is used, the logical record length field is three bytes long for logical records that must be spanned. The DCB LRECL value specifies the maximum logical record length in multiples of 1024.
- It tests whether a spanned record was to have been written.
- If the last record written was not a spanned record, it determines the address of the next buffer segment using the length field of the last record segment moved by the processing program.
- It checks the value placed into the DCBLRECL field to determine if a buffer is ready for emptying and if a new empty buffer is needed. If control is returned from the user and the prior record does not require segmentation (a buffer location is used instead of a record area), the SDW must be changed from the three low-order byte format to the two high-order byte format (0LLL to LL00) when extended logical record interface (XLRI) is used.
- 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 unblocked records are specified, it issues a BALR instruction to pass the present buffer to the EOB 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 tests whether the present record to be written can fit entirely in the new buffer.
- If the record fits, the PUT routine then presents this address to the processing program and returns control to the processing program.
- If the record does not fit, the routine saves the record address in the record area, obtains the address within the record area with the proper alignment, sets the spanned-record flag in the IOB, presents the address in the record area to the processing program, and returns control to the processing program.
- If an EOB condition exists, it tests whether a minimum record segment (at least 5 bytes) can fit in the present buffer.
- If it fits, the routine saves the record address, obtains the address within the record area, sets the spanned-record flag in the IOB, presents the address to the processing program, and returns control to the processing program.
- If it does not fit, the routine issues a BALR instruction to pass the present buffer to the EOB routine. The routine then issues another BALR instruction to obtain a new buffer

through the output-synchronizing-and-error-processing routine, IGG019AR, and determines the address of the first segment of the new buffer. The routine tests whether the present record can fit entirely in the new buffer.

- If a spanned record was to be written out, it restores the record address, determines the length of the segment that can fit in this buffer, moves the segment from the record area to the buffer, and sets the proper flags for the segment.
- If more segments are required, the routine issues a BALR instruction to pass the present buffer to the EOB 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. It moves the remaining bytes of data from the record area to the buffer and sets the proper flags for the segment. This step continues until the entire spanned record has been segmented. The routine then turns off the spanned-record flag and determines the address of the next buffer segment.

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.

When a TRUNC macro instruction is issued after a spanned record was written, this routine branches to the PUT routine to write out the last record (the spanned record) and then truncates the block that contains the last segment of that spanned record.

If a spanned record is being truncated in extended logical record interface (XLRI) mode, the truncate return is set up as if a buffer location, instead of the record area, is being returned to the user.

PUT Module IGG019DJ (SYSIN/SYSOUT): Module IGG019DJ interfaces with a JES to pass the present record into the system output stream. For locate mode, it presents the processing program with the address of the next available buffer segment.

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

Output (SYSOUT specified on the DD statement)

and the DCB specifies:

PUT, PUTX

Simple buffering

Locate, move, or data operating mode

Fixed, undefined, or variable-length record format

Spanned records

Logical record interface

The module consists of PUT, PUTX, and TRUNC macro instructions. (See Diagram M for an overview of the SAM-SI processing for QSAM.) The GET routine is also in this module. It is described in the section on simple-buffering GET routines (see Figure 1 on page 6).

The PUT routine operates as follows:

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

- It determines the type of PUT request and performs the RPL initialization necessary to make the translation to a JES PUT request.

- The record address is placed in RPLAREA and the length of the record is placed in RPLRLEN.

  For move mode the record address is obtained from register 0 on entry to the PUT routine. For locate mode, RPLAREA was set up on the previous invocation of the PUT routine.

  For all record formats other than variable-type, record length is determined by DCBLRECL. For variable format, the current RDW specifies the record size, unless data mode for variable-length spanned records is requested, in which case DCBPRECL contains the record length. Also for variable format, the RDW is excluded from the output record by adjusting RPLAREA past the RDW and decreasing the record length by 4.

Record Format	Value of RPLRLEN
Variable-length record format (move or locate mode)	RDW Length - 4
Variable-length record format (spanned records, locate mode)	value equals total length of all segments in a logical record
Variable-length spanned record format (move mode)	RDW length - 4
Variable-length spanned record format (data mode)	DCBPRECL
Fixed and undefined-length record format (move or locate mode)	DBLRECL

- If processing is in locate mode with variable-length spanned record format, the present segment is moved to the record area. If the SDW indicates the logical record is not complete, the address for the next segment is loaded into register 1 and control is returned to the processing program.

- If the DCB record format indicates ASA or machine control characters, then the control character is checked to determine if it is a Composed Page Data Stream control byte. In this case, the ACB data stream indicator is set (ACBCCDSI) before passing control to the job entry subsystem (JES).

- It passes control to the job entry subsystem (JES) for data transfer by issuing a PUT macro instruction against the RPL. The return code in register 15 is tested upon return from the JES.

- If a control character is indicated in the DCBRECFM field of the DCB, the RPLAREA pointer to the record will be adjusted to point past the control character and the RPLRLEN will be reduced by 1. The address of the control character is placed in the RPLCCHAR field.

  Upon return, register 15 and the RPLRTNCD and RPLCNDCD fields are tested.

- If an error condition is detected, control is passed to the error-processing routine, IGG019AH. (See Figure 12 on page 67.)

- For normal completion, the address in the RPLAREA field is placed in register 1 for locate mode. The RPLAREA field contains the address of the next available buffer.

Registers are restored and control is returned to the processing program.

The PUTX routine operates as follows:

- It receives control when a PUTX macro instruction is encountered in the processing program. This routine processes only the output mode of the PUTX macro instruction.
- The address of the input buffer to be written is located through the DCBRECAD field of the input DCB.
- After having located the output record, the request is then processed by the PUT routine as a PUT, move mode request.

The TRUNC routine receives control when a CNTRL or TRUNC macro instruction is issued. Module IGG019DJ does no processing for these macro instructions. Control is returned to the processing program by IGG019DJ.

PUT Module IGG019FG: Module IGG019FG moves the data portion of the 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

Data operating mode

Variable-length (unblocked or blocked) record format, spanned

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

The PUT routine operates as follows:

- It receives control when the processing program issues a PUT macro instruction.
- It determines the possible location of the next buffer segment by adding the length of the previous record or record segment to the previous buffer segment address. This address is in the DCBRECAD field.
- It then compares the length of the next record with the remaining buffer capacity.
- If ISO/ANSI/FIPS spanned records are being processed, the buffer position pointer is updated to allow room for the 5-byte ISO/ANSI/FIPS segment control word (SCW).
- If the record will fit, the routine moves the record, updates the length field of the block descriptor word (BDW), and checks for blocked records.
- If blocked records are specified, the routine returns control to the processing program. If unblocked records are specified, the routine issues a BALR instruction to pass the current buffer to the EOB routine. The PUT routine issues another BALR instruction to obtain a new buffer through the output-synchronizing-and-error-processing routine, module IGG019AR. The PUT routine then builds a new block descriptor word (BDW) and returns control to the processing program.
- If the record will not fit, the routine determines whether there are 5 or more unused bytes remaining in the buffer.

If there are, the PUT routine breaks the current record so that the first segment fills the buffer. The remaining segment will be placed in subsequent buffers. The length field in the segment descriptor word (SDW) of the first segment is updated to reflect the length of the segment. The third byte of this SDW is set to X'01' to indicate that this segment is the first of a multisegment record. After writing the buffer, the PUT routine does not return control to the processing program until the entire record has been processed. The routine forms the remainder of the current record into a new segment. The new segment is constructed in a new buffer; the third byte of the SDW of the newly created segment is set to X'02' if this segment is the last of a multisegment record. If there are other segments, the third byte is set to X'03' to indicate that this segment is neither the first nor the last of a multisegment record. Newly created segments are processed as any other record.

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.

PUT Module IGG019FJ: Module IGG019FJ 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

Variable-length (unblocked or blocked) record format, spanned

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

The PUT routine operates as follows:

- It receives control when the processing program issues a PUT macro instruction.
- It determines the address of the next buffer segment by adding the address of the last record or record segment moved to the buffer and the length of that record or record segment. The length of the record segment is in the SDW.
- It checks the buffer to see if there are 5 or more unused bytes.
- If there are 5 or more unused bytes remaining in the buffer, the PUT routine places their address into register 1 for the processing program. The PUT routine places the exact number of bytes left in the buffer into register 0 for the processing program. The PUT routine then returns control to the processing program.

- If the buffer contains fewer than 5 unused bytes, the routine issues a BALR to the EOB 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 builds a new block descriptor word (BDW) 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 IGG019FL: Module IGG019FL moves the current 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, spanned

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

The PUT routine operates as follows:

- It receives control when the processing program issues a PUT macro instruction.
- It determines the possible location of the next buffer segment by adding the length of the previous record or record segment to the previous buffer segment address. This address is in the DCBRECAD field.
- It then compares the length of the next record with the remaining buffer capacity.
- If the record will fit, the routine moves the record, updates the length field of the block descriptor word (BDW), and checks for blocked records.
- If ISO/ANSI/FIPS spanned records are being processed, the buffer position pointer is updated to allow room for the 5-byte ISO/ANSI/FIPS segment control word (SCW).
- If blocked records are specified, the routine returns control to the processing program. If unblocked records are specified, the routine issues a BALR instruction to pass the current buffer to the EOB routine. The PUT routine issues another BALR instruction to obtain a new buffer through the output-synchronizing-and-error-processing routine, module IGG019AR. The PUT routine then builds a new block descriptor word (BDW) and returns control to the processing program.
- If the record will not fit, the routine determines whether there are 5 or more unused bytes remaining in the buffer. If there are, the PUT routine breaks the current record so that the first segment fills the buffer. The remaining segment is placed in subsequent buffers. The length field in the segment descriptor word (SDW) of the first segment is updated to reflect the length of the segment. The third byte of this SDW is set to X'01' to indicate that this segment is the first of a multisegment record. After writing the buffer, the PUT routine does not return control

to the processing program until the entire record has been processed. The routine forms the remainder of the current record into a new segment, which is constructed in a new buffer. The third byte of the SDW of the newly created segment is set to X'02' if this segment is the last of a multisegment record. If there are other segments, the third byte is set to X'03' to indicate that this segment is neither the first nor the last of a multisegment record. Newly created segments are processed as any other record.

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.

Update Mode PUTX Routines

The update mode PUTX routines differ from other PUT routines in that PUTX routines share buffers (as well as the DCB and the IOBs) with the update mode GET routines. It is the update mode GET routines that determine the address of the segment, when the end of the buffer is reached and a new buffer is needed. Thus, all that remains for the PUTX routines is to flag the block for output.

There are two update-mode PUT routines. They are part of modules IGG019AE and IGG019BN, which are described under "Update-Mode GET Modules" (see Figure 3 on page 21).

END-OF-BLOCK ROUTINES

The end-of-block routines 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 is specified, two end-of-block routines may be required. When user-totaling is specified, a special user-totaling routine is executed in conjunction with one of the end-of-block routines.

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).

End-of-block routines are shared by BSAM and QSAM. QSAM flow of control is shown in Diagram F; BSAM flow is shown in Diagram G. Register usage at entry to and exit from end-of-block routines is as follows:

Registers	Entry Value	Exit Value
0-1	N/A	Not restored
2	DCB address	Unchanged or restored
3	IOB - 8 (or ICB)	Unchanged or restored

Registers	Entry Value	Exit Value
4-6	N/A	Not restored
7	READ or WRITE CCW offset	Unchanged or restored
8	Caller's base address	Unchanged or restored
9-10	User's registers	Restored[1]
11-12	User's registers	Unchanged or restored
13	Save area	Unchanged or restored
14	Caller's return address	Unchanged or restored
15	Entry point address	Not restored

Note:

[1] These registers are saved by end-of-block in the last two words of the save area, and are restored before returning to caller.

Control passes from an end-of-block routine, through the EXCP or EXCPVR interface, to the I/O supervisor, except when one channel program or IOB is chained to another. End-of-block routines provide device-oriented entries for the channel program, such as control characters and auxiliary storage addresses.

If the American National Standard Code for Information Interchange (ASCII) is used, routines IGG019CC and IGG019CW issue an XLATE macro instruction which translates the entire buffer from EBCDIC to ASCII before writing the buffer. If format-D records are specified, the record descriptor words are converted from binary form to decimal form prior to translation.

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 for tape and unit record devices. The user-totaling routine is described in this section. It moves the contents of the user's totaling area to the user-totaling save area pointed to by the DEB.
- 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 for tape and unit record devices.
- DASD end-of-block routines. These routines perform direct-access device processing for output data sets. The routines attempt to chain IOBs to a queue for which a real-address channel program will be dynamically built by the DASD SIO/pagefix appendage.

Ordinary End-of-Block Routines

Ordinary end-of-block routines process channel programs for tape and unit record devices. This processing is independent of the progress of a previous channel program and causes access to proceed one channel program at a time. For unit-record devices, these routines process control characters and PRTOV macro instructions.

Figure 5 on page 41 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, 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.

End-of-Block Module IGG019CC: Module IGG019CC causes a channel program to be scheduled.

If ASCII coding is used, the entire output buffer is translated from EBCDIC to ASCII. If the ISO/ANSI/FIPS spanned record format (DS/DBS) is used, the 4-byte IBM segment descriptor word (SDW) is converted to the 5-byte ISO/ANSI/FIPS segment control word (SCW) before translation.

The OPEN executor selects and loads this module if the following condition exists:

The DCB specifies normal channel-program scheduling and magnetic tape, card reader, or paper tape as the device type.

The module operates as follows:

- It receives control when a GET or 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, record format is variable, control is received from a PUT or WRITE routine, and a check is made to see if at least 18 bytes are to be written. If not, the record is padded with binary zeros up to 18 bytes or block size, whichever is less; however, with the ASCII feature, format-D records are padded with the ASCII padding character, X'5F', instead of with zeros. An EXCP macro instruction is issued, and control is returned to the PUT or WRITE routine.
- If the device type is magnetic tape and either the record format is not variable or control is not gained from a PUT or WRITE routine, an EXCP macro instruction is issued and control is returned to the GET, PUT, READ, or WRITE routine.
- If an IBM 3525 Card Punch associated data set is being used, a test is made to determine the status of the read-sequence flag.
  - If the read-sequence flag (DCBQSWS field) is on and the associated data set is not READ and print, a WTP message, which indicates that either the GET or READ sequence is invalid, is issued. An abend (003) is issued with a return code of 01. If the read-sequence flag is off, the macro sequence is assumed to be valid and the READ-sequence flag is turned on.
  - Tests are made to determine if the associated data set is either read, punch, and print, or read and punch.
  - If either read, punch, and print, or read and punch is specified in the FUNC parameter, a test is made to determine the status of the punch-sequence flag. If the punch-sequence flag (DCBQSWS field) is on, it is turned off. (This indicates to modules IGG019CE and IGG019CF that their calling routine is in the proper sequence.)
  - If the associated data set is not read, punch, and print, or read and punch, it is assumed that read and print is being used.
  - A test is made to determine the status of the print-sequence flag (DCBQSWS).
  - If the print-sequence flag is on, it is assumed that the print command has been issued. It is turned off so that proper sequencing may continue. If the print-sequence flag is off, it is assumed that the print command has not been issued.

Access Method Options	Selections														
Normal channel program scheduling	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Output, or INOUT OUTIN											X				
Card reader	X	X													
Printer or card punch					X	X	X	X	X	X		X			
Printer (3535)													X	X	
Interpreter/Punch (3525)															X
Data Processing Image (3525)												X			
Magnetic Tape			X	X											
No control character						X		X							
Machine control character						X		X							
ANS control character									X	X					
PRTOV—No user exit					X	X			X						
Label=(,,,IN) or Label=(,,,OUT) on DD statement[1]											X				
User totaling facility				X											
Associated data set		X					X	X		X		X		X	
End of Block Modules															
IGG019AX				AX											
IGG019CC	CC	CC	CC												
IGG019CE					CE	CE	CE	CE							
IGG019CF									CF	CF					
IGG019CT[1]											CT				
IGG019FK												FK			
IGG019FQ													FQ	FQ	
IGG019FU															FU
IGG019TC				TC											

Figure 5. Module Selector—Ordinary End-of-Block Modules (non-DASD)

Note to Figure 5:

[1] When either of these LABEL parameters is specified and the data set is opened for INOUT or OUTIN, the OPEN executor loads module IGG019CT in addition to one of the other end-of-block routines.

End-of-Block Module IGG019CE: Module IGG019CE, if necessary, modifies channel programs for unit record output devices when American National Standard Institute (ANSI) control characters are not used. 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

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).
- If the device is an IBM 3800 Printing Subsystem and OPTCD=J is specified, the module determines if the table reference character in the current record refers to the translate table presently active in the device. If so, the select translate table CCW, which precedes the WRITE CCW, is altered to a NOP. Otherwise, the select CCW is modified to select the appropriate translate table. (If OPTCD=J is not specified, the common printer channel program is used.)
- 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.
- If an associated data set is being used, a test is made to determine the status of the punch-sequence flag.
  - If the punch-sequence flag (DCBQSWS) is on and the associated data set is not punch and print, a WTP message is issued which indicates that either the PUT or WRITE sequence is invalid. An abend (003) is issued with a return code of 02. If the punch-sequence flag is off, the macro sequence is assumed to be valid and the punch-sequence flag is turned on.
  - A test is made to determine if the associated data set is read, punch, and print. If read, punch, and print is specified in the FUNC parameter, a test is made to determine the status of the read-sequence flag.
  - If the read-sequence flag is on, it is turned off. This allows proper sequencing to continue. If the read-sequence flag is off, an ABEND is issued.
  - A test is made to determine the status of the print-sequence flag.
  - If the print-sequence flag is on, proper sequencing continues. If it is off, modules IGG019CE and IGG019CF continue with their normal functions.
  - If the associated data set is punch and print, the status of the print-sequence flag is determined as previously explained for module IGG019CC.

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

End-of-Block Module IGG019CF: Module IGG019CF modifies channel programs for unit record output devices when an American National Standard Institute (ANSI) 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

ANS 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 () which precedes the WRITE CCW (or the select CCW, if the device is a 3800 Printing Subsystem with OPTCD=J specified.)
- If the device is a 3800 Printing Subsystem and OPTCD=J is specified, the module determines if the table reference character in the current record refers to the translate table presently active in the device. If so, the select translate table CCW, which precedes the WRITE CCW, is altered to a NOP. Otherwise, the select CCW is modified to select the appropriate translate table. (If OPTCD=J is not specified, the common printer channel program is used.)
- 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.
- If an associated data set is being used, a test is made to determine the status of the punch-sequence flag.
  - If the punch-sequence flag (DCBQSWS) is on and the associated data set is not punch and print, a WTP message is issued to indicate that either the PUT or the WRITE sequence is invalid. An abend (003) is issued with a return code of 02. If the punch-sequence flag is off, the macro sequence is assumed to be valid and the punch-sequence flag is turned on.
  - A test is made to determine if the associated data set is read, punch, and print. If read, punch, and print is specified in the FUNC parameter, a test is made to determine the status of the read-sequence flag.
  - If the read-sequence flag (DCBQSWS) is on, it is turned off. This allows proper sequencing to continue. If the read-sequence flag is off, an ABEND is issued.
  - A test is made to determine the status of the print-sequence flag (DCBQSWS).
  - If the print-sequence flag is on, proper sequencing continues. If it is off, modules IGG019CE and IGG019CF continue with their normal functions.

- If the associated data set is punch and print, the status of the print-sequence flag is determined, as previously explained for module IGG019CC.

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

End-of-Block Module IGG019CT: Module IGG019CT sets error indicators in the user's DCB and IOB. The OPEN executor selects and loads this module if the following conditions exist:

The data set is opened for INOUT and the DD statement specifies LABEL=(,,,IN)

or

The data set is opened for OUTIN and the DD statement specified LABEL=(,,,OUT)

The module operates as follows:

- It receives control and sets error indicators in the user's DCB and IOB when either of the following conditions exists:

  The DD statement specifies LABEL=(,,,IN), the data set is opened for INOUT, and a WRITE macro instruction is issued,

  The DD statement specifies LABEL=(,,,OUT), the data set is opened for OUTIN, and a READ macro instruction is issued.

End-of-Block Module IGG019FK: Module IGG019FK causes a channel program to be scheduled. The OPEN executor selects and loads this module, if the following conditions are described in the DCB:

Data protection image (DPI) is specified for the 3525 with a read and punch, or read, punch, and print file with normal channel-program scheduling.

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.

- If the READ associated data set has been opened, a test is made to determine the status of the read-sequence flag.

- If the READ associated data set has not been opened, or if the READ-sequence flag is off, a WTP message is issued which indicates that the sequence is invalid. An abend (003) is then issued with a return code of 02. If the read-sequence flag is on (indicating proper sequencing), it is turned off.

- A test is then made to determine the status of the punch-sequence flag (DCBQSWS field). If the punch-sequence flag is on, a WTP message is issued, followed by an ABEND (003). If the punch-sequence flag is off, it is turned on so that proper sequencing may continue.

- It then establishes the buffer area (for the punch operation) according to the format of the data protection image. If a byte in the DPI is blank (X'40'), the module blanks out the corresponding byte in the output punch buffer. If the byte is not blank, the output buffer is not altered. Both areas are 80 bytes in length.

- It returns control to either the PUT or WRITE routine that called it.

End-of-Block Module IGG019FQ: Module IGG019FQ causes a channel program to be scheduled to the 3525 Card Punch. The OPEN executor selects and loads this module, if the following conditions exist:

A print; read, punch, and print; read and print; or punch and print file is specified for the 3525 with either a machine control character, an ANSI control character, or no control character at all with normal channel-program scheduling.

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.
- If either a read, punch, and print or punch and print associated data set has been specified, a test is made to determine the status of the print sequence flag. If the print-sequence flag is on, the CCW pointer is modified to point to the print CCW.
- If both the print- and punch-sequence flags are off, a WTP message is issued to indicate that the sequence is invalid. An abend (003) is then issued with a return code of 03.
- If the print-sequence flag is off, but the punch-sequence flag is on, the module locates the punch DCB and turns off the punch-sequence flag. The CCW pointer is then modified to point to the print CCW and the print-sequence flag is turned on.
- If a read and print associated data set is specified and the print-sequence flag is on, the CCW pointer is modified to point to the print CCW.
- If the print-sequence flag is off, but the read-sequence flag is on, the READ DCB is located and the read-sequence flag is turned off. The CCW pointer is then modified to point to the print CCW and the print-sequence flag is turned on.
- After sequence checking is completed, the module tests for ANSI and machine control characters. If ANSI is specified, the control character is analyzed to determine which line the data is to be printed on. An OR operation is then performed on that line number and the print CCW.
- If ANSI control characters are not specified, the module tests for record format and machine control characters. If machine control characters are specified, they are inserted into the CCW and the buffer address is increased by one.
- If no control character is specified, and two-line printing is specified in the FUNC parameter, the module tests to determine line positioning on the card. This is reflected in the operation code of the print CCW.
- If no control character is specified, and multiline printing is specified, tests are again made to determine line positioning. (Output lines are printed on successive lines.)
- If no control characters are specified, or if they are specified and have been processed, or if either two-line or multiline positioning is complete, the module establishes the WRITE CCW and stores the start address of the CCW for the input/output supervisor (IOS).
- If the PRTOV macro instruction is specified, a check is made for either channel 9 or 12 (depending on which channel is specified in the PRTOV macro instruction).

- The channel program is then executed and a WAIT command is issued. It returns control (via register 14) to either the PUT or WRITE routine that called it.

End-of-Block Module IGG019FU: Module IGG019FU causes a channel program to be scheduled. The OPEN executor selects and loads this module if one of the following conditions exists:

INTERPRET PUNCH is specified for the 3525 with normal channel-program scheduling.

INTERPRET PUNCH is specified for the 3525 with first control character for stacker selection or with no control character at all.

The module operates as follows:

- It retrieves the data address from the WRITE CCW.
- It tests for record format to determine if machine control characters or ANS control characters are being used.
- If either machine or ANS control characters are being used, the data address is increased by one and the control character is inserted into the command byte of the WRITE CCW.
- If machine control characters are not specified, the data address remains unchanged.
- The module blanks out a print buffer. (The print buffer is a 64-byte area located 64 bytes past the beginning of the IOB.) It then moves the final 16 characters of the output punch buffer into the last 16 bytes of the print buffer.
- The channel program start address is stored in the IOB.
- The channel program is then scheduled for execution.
- It returns control (via register 14) to either the PUT or WRITE routine that called it.

End-of-Block Module IGG019TC: The OPEN executor selects and loads this module if the user specified the user-totaling facility (that is, if bit 6 is 1 in DCBOPTCD) for the data set and if the following condition exists:

The DCB specifies normal channel-program scheduling and magnetic tape as the device type.

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.
- The module issues an EXCP macro instruction and returns control to the PUT or WRITE routine.
- It issues a BALR instruction to the user-totaling save routine, IGG019AX, to place the user's total in the user-totaling save area, which is pointed to by the DEB.

User-Totaling Save Module IGG019AX: Module IGG019AX saves an image of the user's totaling area in the sequential access method totaling save area. This save area is described in Figure 36 on page 244.

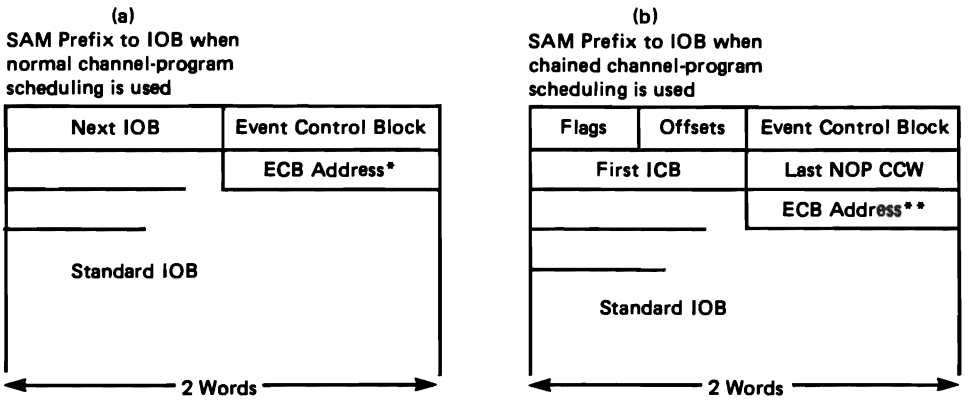
The OPEN executor selects and loads this module if the user-totaling option is specified in the DCB (that is, if bit 6 is 1 in the DCBOPTCD field).

The module operates as follows:

- It receives control from one of the end-of-block routines—IGG019TC, IGG019TV, IGG019TW, or IGG019T2.
- It retrieves the address of the sequential access method totaling save area from the access method portion of the DEB.
- The sequential access method totaling save area contains a pointer to the user's totaling area. An image of the user's total is saved in the next available segment of the sequential access method totaling save area. Then the save area control block is updated so that the pointer identifies the current entry.
- It returns control to the end-of-block routine that called it.

Chained Channel-Program Scheduling End-of-Block Routines (Non-DASD Only)

Chained channel-program scheduling consists of joining the channel programs before execution and disconnecting and posting the channel programs after execution. Joining is performed by the end-of-block routines; disconnecting and posting is performed by appendages. (For a description of the disconnecting process, refer to the program controlled interruption (PCI) and channel-end 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 Figure 8 on page 49.



*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. 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 main IOB's IOBCNOPA field to determine whether to join the channel programs or to issue an EXCP. The routine tries to add to the chain by using CS to test whether the high-order bit of IOBCNOPA is on. If the bit is on, the chain is no longer running. If it is off, the CS instruction changes the last NOP pointer to join the new channel program. CS is used to prevent MP systems from starting two chains at once.

The chained scheduling end-of-block routines, like the ordinary end-of-block routines, provide device-oriented entries for channel programs. For unit-record devices they process control characters. (No processing is performed for the PRTOV macro instruction because it and chained scheduling are mutually exclusive.) There are four chained scheduling end-of-block routines, each of which performs joining and channel program entry processing for a different set of access condition options. Figure 7 lists the available routines 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.

Figure 7 shows that, when chained scheduling is used, the OPEN mode is input, the device type is magnetic tape, and routine IGG019CW is selected and loaded for use as the end-of-block routine for the DCB.

Access Method Options	Selections						
Chained channel program scheduling	X	X	X	X	X	X	X
Input, or	X	X					
Output			X	X	X	X	X
Card reader	X						
Printer or card punch					X	X	X
Magnetic tape		X	X	X			
No control character					X		
Machine control character						X	
ANS control character							X

Figure 7 (Part 1 of 2). Module Selector—Chained Channel-Program Scheduling, End-of-Block Modules—Non-DASD

Access Method Options	Selections								
User-totaling facility				X					
End-of-Block Modules									
IGG019AX[1]				AX					
IGG019CW	CW	CW	CW						
IGG019CX					CX	CX			
IGG019CY									CY
IGG019TW				TW					

Figure 7 (Part 2 of 2). Module Selector—Chained Channel-Program Scheduling, End-of-Block Modules—Non-DASD

Note to Figure 7:

[1] This module is described earlier in this section under "Ordinary End-of-Block Processing."

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 IOBCICBA field	Field does not exist	Address of the first ICB
Contents of IOBCNOPA field	Field does not exist	Address of NOP CCW of last scheduled channel program. The high-order bit is on when the chain is running.

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

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. If ASCII is used, the entire output buffer is translated from EBCDIC to ASCII. If the ISO/ANSI/FIPS spanned record format (DS/DBS) is being processed, the 4-byte IBM segment descriptor word (SDW) is converted to the 5-byte ISO/ANSI/FIPS segment control word (SCW) before translation. The OPEN executor selects and loads this module if one of the following conditions exists:

- The OPEN parameter list specifies input and the DCB specifies chained channel-program scheduling and any device except DASD.
- 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 PUT routine when the 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.
- If the device is magnetic tape, the record format is variable, and control is received from a PUT or WRITE routine, a check is made to see if at least 18 bytes are to be written. If not, the record is padded with binary zeros up to 18 bytes or block size, whichever is less; however, with the ASCII feature, format-D records are padded with the ASCII padding character, X'5F', instead of zeros.
- 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 by doing a CS on IOBCNOPA
- If the joining (the CS) was successful, the routine returns control to the calling routine.
- If 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.

End-of-Block Module IGG019CX: Module IGG019CX, if necessary, modifies channel programs for unit-record output devices when ANS 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 or machine control character

The module operates as follows:

- It receives control from a PUT routine when the 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).
- If the device is a 3800 Printing Subsystem and OPTCD=J is specified, the module determines if the table reference character in the current record refers to the translate table presently active in the device. If so, the select translate table CCW, which precedes the WRITE CCW, is altered to a NOP. Otherwise, the select CCW is modified to select the appropriate translate table. (If OPTCD=J is not specified, the common printer channel program is used.)
- 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 by a CS instruction on IOBCNOPA
- If the joining (the CS instruction) was successful, the routine returns control to the calling routine.
- If 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.

End-of-Block Module IGG019CY: Module IGG019CY modifies channel programs for unit record output devices when ANS 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
- ANS control character

The module operates as follows:

- It receives control from a PUT routine that finds 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 translates the control character and inserts it as the command byte of the control CCW which precedes the WRITE CCW (or the select CCW, if the device is a 3800 Printing Subsystem with OPTCD=J specified.)
- If the device is a 3800 Printing Subsystem and OPTCD=J is specified, the module determines if the table reference character in the current record refers to the translate table presently active in the device. If so, the select translate table CCW, which precedes the WRITE CCW, is altered to a NOP. Otherwise, the select CCW is modified to select the appropriate translate table. (If OPTCD=J is not specified, the common printer channel program is used.)
- 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, using the CS instruction on IOBCNOPA
- If the joining (the CS instruction) was successful, the routine returns control to the calling routine.
- The routine tests the ICB for the present channel program to find whether the joining was successful or not.
- If 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.

End-of-Block Module IGG019TW: Module IGG019TW 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 the user specifies the user-totaling option (that is, if bit 6 is 1 in DCBOPTCD) for the data set and if the following condition exists:

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 PUT routine when the routine finds that a buffer is ready to be scheduled, or from a WRITE routine at the conclusion of its processing.
- It issues a BALR instruction to the user-totaling save routine, IGG019AX, to place the user's total in the user-totaling save area, which is pointed to by the DEB.
- 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 using a CS instruction on IOBCNOPA.
  - If the joining (the CS instruction) was successful, the routine returns control to the calling routine.
  - If 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.

End-of-Block Routines for Direct-Access Storage

For an output request, the end-of-block modules maintain the track balance and calculate the address of the record to be written (that is, the CCHHR address on the direct-access storage device).

The DASD end-of-block modules (see Figure 9 on page 55) process the IOB passed to them by the caller. IOBs built by the DASD OPEN executor contain an IOB extension (IOBEX) with one or two CCWs and other data. The IOB is processed by first chaining it to a queue of active IOBs, then by constructing a real-address channel program that serves the IOB's request, and, finally, by disconnecting and posting the IOB. The DASD end-of-block modules chain the IOB to the active queue. (See "Start I/O (SIO) Appendages" for a description of the channel-program building process; see "Channel End Appendages and Abnormal End Appendages" for a description of the disconnecting and posting process.)

A queue of IOBs is made active by storing the address of the queue's first and last IOBs in the interrupt control queue element (ICQE), in fields ICQFIRST and ICQENDA, and issuing an EXCPVR SVC. The DCB contains the ICQE's address (at DCBICQE or DCBIOBAD[1]). The ICQE is built by the DASD OPEN executor.

The IOB in the sequential access method block (SAMB) is passed to EXCPVR. (**Note:** The IOB in the SAMB is not part of the active IOB queue.) The IOB in the SAMB is pointed to by the ICQE (ICQIOBA). The SAMB is built by the DASD OPEN executor.

The DASD end-of-block modules either chain the IOB passed to them to an active IOB queue, or issue an EXCPVR SVC. The compare-and-swap (CS) instruction is used to attempt to update ICQENDA with the address of the IOB. The CS instruction tests the high-order bit of ICQENDA (ICQEXND).

If ICQEXND is zero, the swap is successful: The active queue of IOBs is updated to include another IOB. The end-of-block module then returns to the caller.

If ICQEXND is one, no active IOB queue exists and the swap fails. The IOB's address is put into ICQFIRST and ICQENDA, and an EXCPVR SVC is issued. The end-of-block module then returns to the caller.

DASD end-of-block modules are loaded for all BSAM, BPAM, and QSAM direct-access processing, except for BFTEK=R processing (see "READ Module IGG019BU") and for WRITE-load processing (that is, BDAM create processing). IGG019TV is the end-of-block module for all processing except track overflow output. IGG019T2 is loaded for track overflow output.

---

[1] DCBICQE and DCBIOBAD are labels for the same DCB field.

Access Method Options	Selections								
Input or update	X								
INOUT or OUTIN		X	X		X	X			
Output							X		X X
Track overflow			X						X
LABEL=(,,,IN) or LABEL=(,,,OUT) on a DD statement[1]					X				
User totaling						X			X
Direct-access	X	X	X		X	X	X		X X
DASD End-of-Block Routines									
IGG019AX[2]						AX			AX
IGG019CT[1,2]					CT				
IGG019TV	TV	TV	TV				TV		
IGG019T2			T2					T2	

Figure 9. Module Selector—DASD End-of-Block Routines

Notes to Figure 9:

1 When either LABEL=(,,,IN) and OPEN for INOUT or LABEL=(,,,OUT) and OPEN for OUTIN is specified, IGG019CT is loaded in addition to one of the other end-of-block routines.

2 This module is described in "Ordinary End-of-Block Routines."

END-OF-BLOCK MODULE IGG019TV: For an output request, module IGG019TV computes a valid storage address (CCHHR) for the data record (using the track balance value and, if necessary, further allocated extents on the volume), and then attempts to chain an IOB to an active IOB queue. For an input request, module IGG019TV attempts to chain an IOB to an active IOB queue. The OPEN executor selects and loads IGG019TV when the DCB specifies:

Direct-access storage

Not track overflow output

The module operates as follows:

- It receives control from a GET or PUT routine that finds a buffer is ready to be scheduled, or from a READ or WRITE routine at the conclusion of its processing.
- If the user specified the user-totaling option (that is, if bit 6 in DCBOPTCD is 1) for the data set, IGG019TV issues a BALR instruction to the user-totaling save routine, IGG019AX, to place the user's total in the user-totaling save area, which is pointed to by the DEB.

For an output request:

- 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 DCBCVTBL field.) 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 the 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 end-of-volume, 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 and 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 IOB extension field (IOBCNT).

For an input request:

- If more than one IOB is associated with the DCB, the module (a) checks for a cylinder change in the IOBSEEK field in the next IOB by comparing it with the cylinder value in the DCBFDAD field in the DCB, and (b) copies the IOBSEEK field in the next IOB into the DCBFDAD field in the DCB.
- If only one IOB is associated with the DCB, the module (a) checks for a cylinder change by comparing the cylinder value in the IOBSEEK field in the IOB with the cylinder value in the DCBFDAD field in the DCB, and (b) copies the CCHHR portion of the DCBFDAD field in the DCB into the CCHHR portion of the IOBSEEK field in the IOB.

For an input request, and for an output request:

- If a change in cylinder value was found for an input request, or if the updated DCBFDAD value indicates record 1 on track 0 for an output request, and in either case the cylinder value is on a page boundary (evenly divisible by 8), the DEBXFLG1 field in the DEB extension is checked for an MSS window processing request. If such processing is indicated, the ICBCHKAR macro is issued to invoke SVC 126, which will relinquish the processed window and acquire the next one. A GETMAIN for a 12-byte SVC 126 parameter list is issued before ICBCHKAR is issued. A related FREEMAIN is issued upon return from SVC 126.
- If the number of requests (ICQNOQ) equals the maximum number of requests (ICQMAXQ), it does the following; otherwise, it increases ICQNOQ by 1 and returns to the caller.
- It uses the compare and swap (CS) instruction to attempt to chain the IOB to the active IOB queue as the last IOB (ICQENDA). If the swap is not successful, the IOB address

is placed in the ICQE (ICQFIRST and ICQENDA), and an EXCPVR SVC is issued.

- After issuing the EXCPVR SVC (or if the swap is successful), the module returns to the caller.

End-of-Block Module IGG019T2: 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 10). 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 built by the SIO/pagefix appendage, IGG019BX, uses the search command with the multiple-track (M/T) mode.

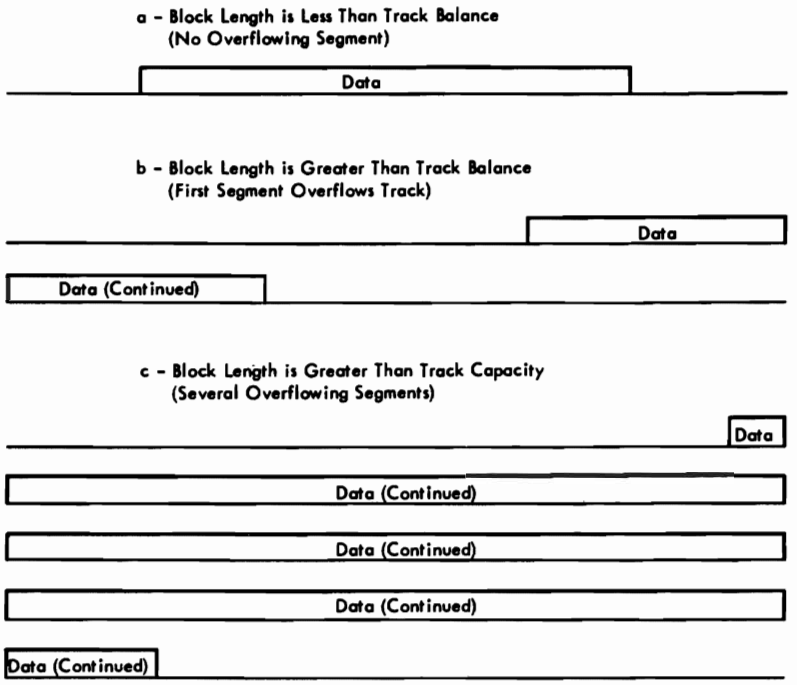


Figure 10. Track-Overflow Records

Module IGG019T2 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 the routine finds that a buffer is to be scheduled, or from a WRITE routine at the conclusion of its processing.

- If the user specifies the user-totaling option for the data set, IGG019T2 issues a BALR instruction to the user-totaling save routine, IGG019AX, to place the user's total in the user-totaling save area, which is pointed to by the DEB.

- It compares the block length with the space remaining on the track last written on.

- It initializes to zero the track overflow data field of the IOB extension, IOBTRKOV. The address of the data block and the length of the entire block have been placed in IOBCCW1 by the WRITE or PUT routine.

- If the entire block fits on this track, the module sets IOBLFST (that is, the length of the first or only overflow segment) to the length of the data block. The module next updates the track balance, and then attempts to add the IOB to the active IOB queue.

  If the updated MBBCCHHR (saved in the ICQE) indicates record 1 on track 0 of a cylinder on a page boundary (evenly divisible by 8), a test is made for MSS window processing. If such processing is indicated in the DEBXFLG1 field of the DEB extension, the ICBCHKAR macro is issued to invoke SVC 126, which will relinquish the processed window and acquire the next one. A GETMAIN for a 12-byte SVC 126 parameter list is issued before ICBCHKAR is issued. A related FREEMAIN is issued upon return from SVC 126.

- If at least one data byte (including the key, if any) fits on this track, the module sets IOBLFST to the key length plus the data length of the segment of the block which fits on the track and tests for another track in the same extent.

- If the next track is in this extent, the module compares the remaining block length with the track capacity.

- Tests are made to determine if MSS window processing is needed. If the updated MBBCCHHR (saved in the ICQE) indicates record 1 on track 0 of a cylinder on a page boundary (evenly divisible by 8), a test is made for MSS window processing. If such processing is indicated in the DEBXFLG1 field of the DEB extension, the ICBCHKAR macro is issued to invoke SVC 126, which will relinquish the processed window and acquire the next one. A GETMAIN for a 12-byte SVC 126 parameter list is issued upon return from SVC 126. The module then proceeds as it does when at least one byte fits on the track.

- If the remainder of the block exceeds the track capacity, the module sets IOBLMID (that is, the length of the middle segment) to the track capacity. The module next increases IOBNMID (that is, the number of middle segments) by 1 and increases IOBNINCL (that is, number of segments written on the cylinder that contains the first segment) by 1 if this segment is to be written on the same cylinder as the first segment. Next, the module determines whether the next track is in this extent.

- If the remainder of the block is less than the track capacity, the module sets IOBLLST (that is, the length of the last overflow segment) to the data length of the last segment. The module next increases IOBNINCL if the last segment is to be written on the cylinder that contains the first segment. Finally, the module updates track balance and attempts to add the IOB to the active IOB queue.

- If the next track is not in this extent, the module sets the CCW command code in IOBCCW1 to "erase" (X'01') and attempts to chain the IOB to the active IOB queue. After either successfully chaining the IOB or issuing an EXCPVR SVC, the module waits for the requests's completion (and for the completion of all previous IOBs on the active IOB queue). Completion is posted in the SAMB IOB's ECB, which is located

at ICQECB. This process erases all unused data tracks at the end of the extent: The track overflow record is too long and cannot fit in the current extent, but must be written in the next extent. After the ICQECB is posted (that is, the queue of IOBs is empty), the module tests for another extent.

- If there is another allocated extent on this volume, the module reconstructs the IOBTRKOV field in the IOB extension by proceeding as it does 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 end-of-volume, and returns control to the PUT or the WRITE routine without attempting to add to the active IOB queue. The EOV condition is eventually recognized and processed—in QSAM by the synchronizing routine, and in BSAM by the CHECK routine.

SYNCHRONIZING-AND-ERROR-PROCESSING ROUTINES

A synchronizing-and-error-processing routine (1) synchronizes execution of the processing program with execution of the channel programs and (2) performs error processing to permit continued access to the data set after an error is encountered during the execution of a channel program. An error-processing routine performs only the latter function.

There are five synchronizing-and-error-processing routines. (See Figure 11 on page 60.) These 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

There are three error-processing routines. (See Figure 12 on page 67.) These routines:

- Are shared by QSAM and BSAM
- Only process errors
- May be either synchronous or asynchronous

The track-overflow, 3203 and 3211 Printer retry error-processing routines are asynchronous. They receive control by being scheduled by an abnormal-end appendage. The SYSIN/SYSOUT error-processing routine is synchronous and receives control directly from a GET or PUT routine (QSAM) or from a CHECK routine BSAM).

In some cases the QSAM synchronizing routines issue an SVC 55 (EOV) to distinguish between permanent error and end-of-volume conditions. For a permanent error, the EOV routine returns control to the synchronizing routine, which in turn passes control to the user's SYNAD routine. If the SYNAD routine returns, the synchronizing routine again invokes EOV to implement error options. For accept and skip, control returns once more to the synchronizing routine. It now operates as when it is first entered.

For an end-of-volume condition (unit exception), EOV takes one of the following actions:

1. It may return to the synchronizing routine with a new DEB, after restarting channel programs. The synchronizing routine then operates as when it is first entered. A new volume is being processed, possibly because of a data set concatenation with like or unlike attributes.

2. It may exit to the user's EODAD routine if the condition should be treated as end-of-file.

3. It will ABEND if unable to take the appropriate action above.

QSAM synchronizing routines have a standardized register usage allowing them to be used interchangeably by GET/PUT routines. This register usage is shown below:

Registers	Entry Value	Exit Value
0-1	N/A	Not restored
2	DCB pointer	Unchanged or restored
3	Previous IOB-8 (or ICB)	New IOB-8 (or ICB)[1]
4	N/A	Unchanged or restored
5	N/A	New buffer address
6	N/A	Unchanged or restored
7	READ or WRITE CCW Offset	Unchanged or restored
8	N/A	Caller's base address[2]
9-12	User's registers	Unchanged or restored
13	Save area[3]	Unchanged or restored
14	Caller's return address	Unchanged or restored
15	Entry point address	Not restored

Notes:

[1] This value also stored in DCBIOBA.

[2] Obtained from save area.

[3] Registers 15-8 must be stored beginning at offset 24 (decimal). This offset is not the standard one used by the system.

The routines described in Figure 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. Figure 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.

Access Method Options	Selections					
GET	X	X			X	X
PUT			X			
Input, Readback		X				
Output			X			
Update	X				X	
Variable-length record format					X	
Spanned records					X	
Locate operating mode					X	
* or DATA specified on DD statement[1]					X	
Modules						
IGG019AF	AF					
IGG019AQ		AQ				AQ
IGG019AR			AR			
IGG019BQ					BQ	

Figure 11. Module Selector—QSAM Synchronizing-and-Error-Processing Modules

Note to Figure 11:

[1] If SYSOUT is specified on the DD statement, none of the synchronizing and error-processing modules are required. The necessary routines are contained within the compatibility interface processing module, IGG019DJ (see Figure 1 on page 6).

Synchronizing Module IGG019AF (Update): Module IGG019AF finds the next buffer and ensures 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 force-end-of-volume (FEOV) macro instruction is encountered in a processing program, once from the update GET routine (when the FEOV routine schedules the last buffer) and once directly from the FEOV routine (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 completed processing, the module issues a WAIT macro instruction.

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

- It receives control when the update GET routine finds that a new buffer is needed or when the FEOV routine awaits execution of the scheduled buffers.
- If the channel program for the next buffer encountered an end-of-volume condition, or if this module has control because of an FEOV macro instruction, the module finds the IOBs flagged for output. It then turns on the write-only flag and schedules the IOB for processing by means of an EXCPVR macro instruction.
- When all IOBs have been processed, or if none are pending, the module passes control to the EOV routine by way of an SVC 55 instruction. If this module has control because of an FEOV macro instruction, control returns to the routine that passed control.
- If a permanent error is encountered for a write-only IOB scheduled 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 IOBs flagged for output are rescheduled for execution using an EXCPVR macro instructions.

  Terminate option: Control passes to the EOV routine to request an ABEND macro instruction.

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

- It receives control when the update GET routine finds 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 indicate read-only and issues an EXCPVR 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 EXCPVR macro instruction for all the IOBs except the present one.

  The module ensures 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 EXCPVR macro instruction for all IOBs.

The module ensures 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, and issues an EXCPVR macro instruction for these empty channel programs. On execution of all the channel programs, the module passes control to the EOV routine to request an ABEND.

Synchronizing Module IGG019AQ (Input): Module IGG019AQ finds the next input buffer, determines its status, and passes a full buffer to the GET routine. If ASCII is used, the entire input buffer is translated from ASCII to EBCDIC. If ISO/ANSI/FIPS spanned record format (DS/DBS) is used, the 5-byte ISO/ANSI/FIPS segment control word (SCW) is converted to the IBM 4-byte segment descriptor word (SDW), which leaves an unused byte at the beginning of each segment.

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

INPUT or RDBACK

or,

INPUT for SYSIN (* or DATA specified on the DD statement)

and the DCB specifies:

GET

The module operates as follows for SYSIN data sets:

- It receives control when the SAM subsystem interface (SAM-SI), QSAM processing module IGG019DJ, detects an end-of-data condition.
- It loads the DCB address into register 1 and issues an EOV SVC 55 instruction. Control is returned to this module only if the SYSIN data set is concatenated to another input data set.
- If control is returned to this module, the EOV close bit is set in the DCBOFLGS field. A test is made to determine if the unlike attribute bit (DCBOFLGS) is set. If it is, control is returned to the processing program. If not, a branch is taken to the GET routine to reschedule the last GET request before returning to the processing program.

If a SYSIN data set was not specified, 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 has not yet completed processing, the module issues a WAIT macro instruction.
- If the channel program has been executed normally, the module uses XLATE if necessary to convert ASCII records to EBCDIC, then updates the DCBIOBA field to point to this IOB, and returns control to the GET routine. If format-D records are being read, the record descriptor words are first converted from decimal to binary code.

- If the channel program has been completed normally, and if the buffer contains a DOS checkpoint record, tape files only, the module 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 EOV routine. EOV returns with a new DEB only if another volume is allocated to the data set or if another input data set is concatenated with it. In that case, EOV has rescheduled the purged channel programs. If EOV returns with a nonzero value in register 15, the DEB has not been changed and the SYNAD routine is to be entered.

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 the output device is a 3203 or 3211 Printer and three or more buffers are being used, the synchronizing module waits for two channel programs to be completed before updating the DCBIOBA field.

- If an error occurred during the execution of the channel program, the module issues an SVC 55 instruction to pass control to the EOV routine. EOV returns with a new DEB only if it is able to allocate another extent or volume to the data set. In that case, EOV has rescheduled the purged channel programs. If EOV returns with a nonzero value in register 15, the DEB has not been changed and the SYNAD routine is to be entered.

Synchronizing Module Module IGG019BQ (Update): Module IGG019BQ finds the next buffer and ensures that it has been refilled. If a unit status prevented refilling of the buffer, the module processes the pending channel programs according to whether they are empty-and-refill or refill-only requests. The OPEN executor selects and loads this module if the OPEN parameter list specifies:

Update

Locate operating mode

and the DCB specifies:

GET

Variable-length spanned (blocked or unblocked) record format

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 an FEOV macro instruction is encountered in a processing program, once from the update GET routine (when the FEOV routine schedules the last buffer) and once directly from the FEOV routine (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 EOV condition is encountered:

- It receives control when the update GET routine finds that a new buffer is needed or when the FEOV routine awaits execution of the scheduled buffers.

- If the next IOB encountered an EOV condition, the module tests whether assembling or updating of a spanned record is in process.

- If updating is in process, the module delays the normal EOV processing by turning off the error flags in the DCB and then returns control to the update GET routine.

- If assembling is in process, the module sets the spanned record flag in the IOB and continues to the next step.

- If assembling is in process or if this module has control because of an FEOV macro instruction, the module finds the IOBs flagged for output. It then sets the write-only flag in the IOB and schedules the empty channel programs for execution by means of an EXCPVR macro instruction.

- If all empty channel programs have been executed, or if none are pending, the module issues an SVC 55 instruction. If this module has control because of 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 EOV condition or for an FEOV macro instruction, control passes to a 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: The pending empty channel programs are rescheduled for execution using EXCPVR macro instructions.

  Terminate: Control passes to the abend routine.

- On return of control from the EOV routine the module tests whether assembling of a spanned record is in process. If it is being processed, the module turns off the spanned-record flag in the IOB and returns control to the update GET routine.

The module operates as follows if a permanent error is 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: If the error occurred in the empty portion of a channel program, the module sets the IOB's read-only flag and issues an EXCPVR 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 EXCPVR macro instruction, which reestablishes all the IOBs on an active queue except the present one.

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

  Skip: 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 treats this as a RELSE condition and issues an EXCPVR macro instruction, which restarts all IOBs on an active queue.

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

  Terminate: If the error option 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 sets the IOB's write-only flag and issues an EXCPVR macro instruction for these empty channel programs. On the execution of all the channel programs, the module passes control to the ABEND routine.

SYSIN/SYSOUT Synchronous-Error-Processing Module IGG019AH: Module IGG019AH is used in both BSAM and QSAM. It processes permanent error conditions detected during the processing of requests for a SYSIN/SYSOUT data set. This routine is loaded by the SAM-SI GET or PUT routine or by a CHECK module when the error is detected, and is entered with a BALR instruction. When IGG019AH returns control to the calling program, the module is deleted.

The module contains an exit routine that is entered from SYNADAF. This routine formats the SYNADAF message. The routine is entered by SYNCH and its address is found in the SVC exit list. It returns control to SYNADAF.

The module also contains a SYNAD control routine. Upon entry, it first checks to see if the user has provided the address of a SYNAD routine in the DCB. If no routine is specified, control is returned to the calling routine (QSAM EROPT=ACC or SKP) or issues an ABEND (BSAM or QSAM EROPT=ABE).

If a SYNAD routine is specified, IGG019AH operates as follows:

- The entry point to the SYNADAF exit is stored in the SVC exit list.
- A dummy IOB is formatted. Parameter registers 0 and 1 are loaded with the IOB address (QSAM) or the DECB address (BSAM), the DCB address, and error flags.
- The user's registers are saved in a new save area which is obtained with a GETMAIN macro instruction.

- The current registers are saved in the user's save area and the user's registers are loaded and the SYNAD routine is entered with a BALR instruction.

If DCB EROPT is ACC or SKP, the user SYNAD routine returns control to IGG019AH, the register save sequence is reversed, the new save area is freed, and control is returned to the calling routine. If EROPT is ABE, problem determination message IEC020 is issued followed by a 001 ABEND.

See Figure 12 for error-processing module selection.

Access Method Options	Modules
3203-4 or 3211 Printer	IGG019FS
*, DATA, or SYSOUT specified on DD statement	IGG019AH

Figure 12. Module Selector—Error-Processing Modules

IBM 3211 Printer Asynchronous-Error-Processing Module IGG019FS (Print Line Buffer Error—Retry): Module IGG019FS is device-dependent and is scheduled asynchronously by the 3211 abnormal-end appendage (IGG019FR, IGG019CU, or IGG019V6). The module retries operations that result in print line buffer parity errors or UCS buffer parity errors, whenever possible. When an operation cannot be retried, the printer is reset and control is returned to the calling program.

The module operates as follows:

- It initializes registers to point to the DCB, ECB, and IOB. It then examines sense bytes in the IOB to determine if one of the error conditions for which a retry is possible occurred.
- If a UCS buffer parity error is indicated (ECB posted in error with an X'41' or X'44' and the command retry bit is on in sense byte 1), the UCS image ID is obtained from the UCB located in SYS1.IMAGELIB and loaded into storage. (Failure to locate the UCS image in the SYS1.IMAGELIB causes a skip to channel 0 command to be issued. This resets the printer and the module returns control to the calling program.) An IOB and channel program to load the UCS image into the UCS buffer on the 3211 are constructed and executed. If a permanent I/O error occurs during an attempt to load the UCS buffer, a skip to channel 0 command is issued to reset the printer. The UCS field in the UCB is also set to 0 and control is returned to the calling program. If the UCS buffer is loaded successfully, a check is made to determine the access method (BSAM or QSAM) being used.
- When QSAM is being used, a check is made to determine if three or more buffers were specified in the BUFNO field of the DCB macro instruction. (This is a condition necessary to retry a print line.) After either UCS buffer parity errors or print line buffer parity errors, the type of scheduling is determined. For normal scheduling, the IOB associated with the failing print line is located and the channel program for that IOB is reissued once. If the channel program is not successful, the next IOB is rescheduled if necessary and control is returned to the problem program, as though no error occurred. If the channel program is not successful, a skip to channel 0 command is issued to reset the control unit and the module returns control to the calling program. For chained channel scheduling, the portion of the channel program associated with the failing print line is reissued. If it is successful, a check is made to determine if another chain

needs to be started before the return to the problem program. If the retry is unsuccessful, a skip to channel 0 command is issued and the module returns control to the calling program.

- For BSAM, or for QSAM with fewer than three buffers specified, a skip to channel 0 command is issued and the module returns control to the calling program.

See Figure 12 on page 67 for error-processing module selection.

APPENDAGES

Appendages are access method routines that receive control from and return control to the I/O supervisor. They operate in the supervisor state. The same appendages are used in QSAM as in BSAM.

An appendage receives control from the I/O supervisor and tests and may alter the channel status word (IOBCSW). The I/O supervisor uses the IOBCSW to post the event control block (ECB). If the SIO appendage receives control from the I/O supervisor before the latter starts execution of the channel program, it may alter channel commands just before channel program execution. The relationship of the I/O supervisor and the appendages is shown in Diagram F.

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; otherwise, control remains with the I/O supervisor.

The I/O supervisor exits where appendages receive control are:

- End-of-extent
- SIO
- Pagefix (offset 4 into the SIO appendage if the DEB indicates a pagefix appendage exists)
- Channel end
- PCI
- Abnormal end

The I/O supervisor unconditionally schedules the routine at the address associated with the exit in the appendage vector table. If no appendage is present, the entry points to an instruction that causes immediate return to the I/O supervisor.

When a VIO data set is processed, the I/O supervisor passes control to the VIO interface routine (IDDWIAPP). The appendages then receive control from the VIO routine with the same interfaces as with the I/O supervisor. When some VIO data sets are processed (that is, BSAM or QSAM, READ or WRITE, not track overflow, not update, and not BSAM create-BDAM data sets), the appendages may be loaded but never get control. A special VIO routine (IDDWISVR) simulates the I/O functions and all the required actions of the appendages. For more details about VIO processing and modules, see VIO Logic.

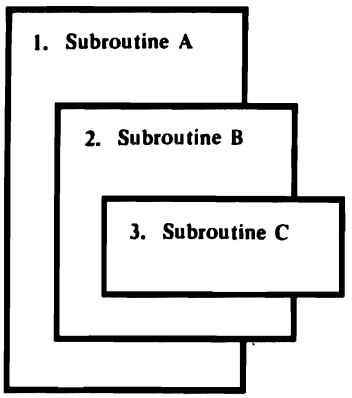
Appendages differ from other sequential access method routines that are loaded by the OPEN executor into processing program virtual storage. They differ because they operate asychronously with the processing program. The events that cause appendages to gain control depend on the progress of the channel program, not on the progress of the processing program. Other appendages operate by running enabled under an SRB.

The OPEN executor selects and loads all the appendages to be used with a DCB. No appendage, one appendage, or several appendages may be used with one DCB. The OPEN executor places the addresses of the required appendages into the various fields of the appendage vector table. Figure 13 on page 71 lists the appendages and the conditions that cause the different appendages to be used. The appendages are grouped according to the condition detected by the I/O supervisor before control is passed to the appendage. Note that some appendages have entry points for more than one of the conditions checked by the I/O supervisor.

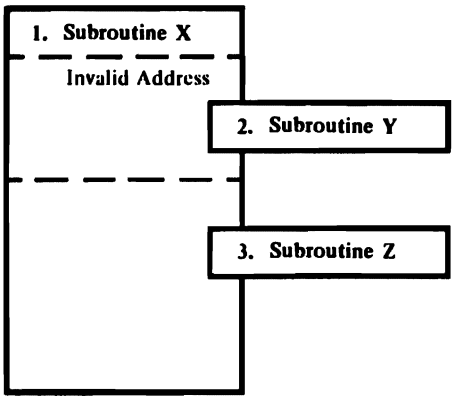
How to Read Compendiums

The compendiums (or hierarchic tables) used to illustrate the following appendages do not usually show all the exits and entrances to a given module. A compendium depicts the flow of control, among subroutines, that relates to a particular function.

Each block in a compendium is associated with a subroutine name. The blocks are nested (indented) to show the sequence of calls. For example, in the following diagram, subroutine "A" at some point in its processing calls subroutine "B," and subroutine "B" at some point in its processing calls subroutine "C." Unless otherwise indicated by an exit indicator, the called subroutines always return control to the calling subroutines in the sequence in which they are called.



In many instances, a call to a subroutine is conditional. In these cases, the condition that must be met is shown in the block that represents the calling subroutine. Dotted lines delineate the calls affected by a given condition. This is illustrated in the following example:



In this example, subroutine "Y" is called when an invalid address is detected, and subroutine "Z" is always called. In either case, subroutine "Y" and "Z" always return control to subroutine "X."

Each subroutine is numbered to key it to the extended description on the page that faces the diagram. The extended description provides details about the conditions that exist when a call is made and about the general processing that is performed by the subroutines.

Start I/O (SIO) Appendages

Start I/O (SIO) appendages, if present, gain processor control when the start I/O subroutine of the EXCP supervisor reaches the start I/O appendage exit. The following appendages set channel program entries:

- 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.
- IGG019BX. This is the SIO/pagefix appendage. It checks the validity of I/O requests, and, during pagefix processing, it passes control to IGG019BY. It is loaded for all direct-access processing except BDAM create.
- IGG019BY. This module processes pagefix requests passed to it by IGG019BX. It builds real-address programs and maintains pagefix lists. It is loaded for all direct-access processing except BDAM create.

  All control blocks and data areas used by the I/O interruption supervisor and appendage modules must be mapped into real storage. If they are not and the I/O interruption supervisor encounters a page exception, the task that requested the I/O is abnormally terminated. The EXCP portion of the I/O supervisor determines that certain control blocks and data areas will be referred to during later processing.

Access Method Options	Selections									
Input, INOUT, OUTIN			X	X					X	X
READBACK			X							
Create BDAM		X								
RECFM=FB			X							
RECFM=V				X					X	
RECFM=VS		X								
DASD	X	X								
Printer				X						
Chained scheduling						X	X			
3211								X		
Magnetic tape (OPTCD=H)									X	X
V=R							X			
Appendages entered from Pagefix Exit:										
IGG019BX[1]	BX									
IGG019BY[1]	BY									
Appendages entered from SIO Exit:										
IGG019BX[1]	BX									
IGG019BY[1]	BY									
IGG019CL				CL						
Appendages entered from Channel End Exit:										
IGG019BT		BT								
IGG019BZ[3]	BZ									
IGG019CI			CI							
IGG019CJ				CJ						
IGG019CU						CU				
IGG019EI									EI	
IGG019EJ										EJ
IGG019V6[2]							V6			
Appendages entered from PCI Exit:										
IGG019V6							V6			
Appendages entered from Abnormal End Exit:										
IGG019BZ[3]	BZ									

Figure 13 (Part 1 of 2). Module Selector—Appendages

Access Method Options	Selections				
IGG019CU[3]		CU			
IGG019EI[3]					
IGG019EJ[3]					EJ
IGG019FR				FR	
IGG019V6[1]			V6		

Figure 13 (Part 2 of 2). Module Selector—Appendages

Notes to Figure 13:

1 The module has multiple entry points. The module is described in "Start I/O (SIO) Appendages."

2 The module has multiple entry points. The module is described in "Program Controlled Interruption Appendages."

3 The module has multiple entry points. The module is described in "Channel-End Appendages."

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:

- It tests to see if the EXCP was issued from an SVC routine. If so, it effects a normal return to EXCP.
- 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 returns control to the EXCP supervisor.

EXCPVR Processing Appendages

Modules IGG019BX and IGG019BY are the DASD SIO/pagefix appendage. They operate in conjunction with the DASD channel-end/abnormal-end appendage (IGG019BZ) and the DASD end-of-block modules (IGG019TV and IGG019T2) to process IOBs on an active IOB queue. The active IOB queue is addressed by the ICQE.

An end-of-block module forms the active IOB queue, addressed by the ICQE, and issues an EXCPVR SVC. For QSAM, ICQMAXQ is set to BUFNO minus 1 with a limit of 29. The end-of-block module passes a group of requests to EXCPVR by queuing the requests until it gets the number of requests equal to ICQMAXQ plus 1. This ensures that the requests are always passed in groups. The EXCPVR processor passes control to the pagefix appendage entry of IGG019BX.

Module IGG019BX passes control to module IGG019BY which constructs a channel program in the SAMB. The channel program serves the requests that are represented by as many IOBs on the active IOB queue as possible. Module IGG019BY also builds a pagefix list in the SAMB for the user buffers referred to by the channel program.

The address of the pagefix list is returned to the EXCPVR processor, which fixes the pages in the list (that is, makes the pages in virtual storage temporarily not movable). The EXCPVR processor then passes control to the SIO appendage entry of IGG019BX. The SIO appendage completes construction of the channel program by replacing the virtual addresses of buffers in the CCWs indirect address word (IDAW) lists built in the SAMB. When the SIO appendage returns to the EXCPVR processor, the real-address channel program is executed. The EXCPVR processor returns to the caller, an end-of-block module.

While the channel program is running, more IOBs might be added to the end of the active IOB queue (addressed by the ICQE) as a result of additional READ, WRITE, GET, and PUT requests against the DCB.

When channel end occurs for the channel program, the EXCPVR processor passes control to the channel-end appendage entry of IGG019BZ. The channel-end appendage posts the ECBs for those IOBs on the active IOB queue whose requests were satisfied by the channel program, and then removes those IOBs from the active IOB queue.

If no errors were encountered and there are no more IOBs on the active IOB queue, the channel-end appendage sets the EXCPVR-required flag (ICQEXND, the high-order bit of ICQENDA in the ICQE) and returns normally to the EXCPVR processor. The EXCPVR processor terminates processing for the EXCPVR SVC, and then returns to the caller.

If there are more IOBs on the active IOB queue, the channel end appendage takes the "re-EXCPVR" return to the EXCPVR processor. The EXCPVR processor passes control to the SIO appendage (IGG019BX), which processes the IOBs that remain on the active IOB queue. (The pagefix appendage is not entered for "re-EXCPVR" processing.)

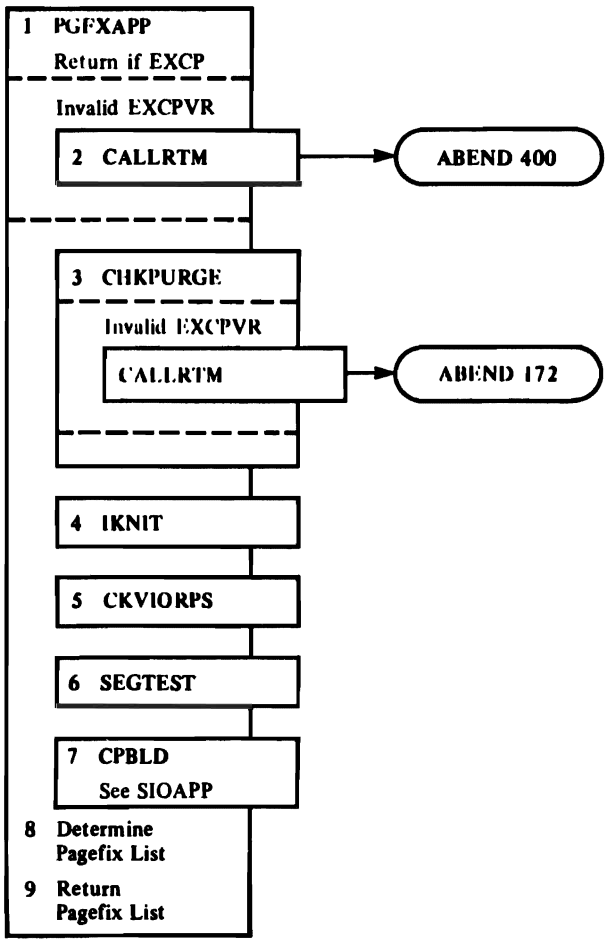
For a "re-EXCPVR" entry, the SIO appendage builds a new channel program to serve as many of the IOBs on the active IOB queue as possible, and builds a pagefix list for the user buffers referred to by the channel program. The pagefix list is compared to the previous list, and the system pagefix and pagefree routine is called as needed: If the two pagefix lists contain only one entry and the entries are identical, no action is taken; otherwise, the old pagefix list is pagefreed, and the new pagefix list is pagefixed. IDAW lists are built for the new channel program. When the SIO appendage returns to the EXCPVR processor, the channel program is executed.

The process of building a channel program to serve some (or all) of the IOBs, executing the channel program, posting and freeing the IOBs served, and building a new channel program to serve additional IOBs might continue indefinitely. The process continues for as long as the user's program schedules IOBs (that is, adds IOBs) to the end of the active IOB queue (addressed by the ICQE).

If the active IOB queue empties, an EXCPVR SVC starts the process again when the user's program makes a new request (that is, issues a READ, WRITE, GET, or PUT macro).

This page intentionally left blank.

Pagefix Appendage (PGFXAPP): This compendium illustrates the pagefix appendage.



Appendage IGG019BX/IGG019BY (SIO/Pagefix)

The OPEN executor selects and loads this appendage for use as the SIO appendage if the DOB specifies direct-access processing.

Module IGG019BX processes SIO requests. If the pagefix entry is used by the calling program, module IGG019BX passes control to module IGG019BY, which completes the pagefix processing. Module IGG019BX processes the SIO requests. Note that these modules share several common subroutines during SIO and pagefix request processing.

The pagefix appendage (PGFXAPP) operates as follows:

1. If flag RQEll4 is on, EXCPVR SVC was issued. If flag RQEll4 is off, an EXCP SVC was issued and is ignored by this appendage. Registers 10 and 11 are set to zero to indicate, on return to EXCP, that no pagefix list needs to be established.

2. If the DEBXSAMB field value is equal to the SAMBREG value, the EXCPVR request is against the SAMB IOB. If the SAMB IOB is not referred to by the EXCPVR request, the CALLRTM macro is issued (with abend code 400).

3. The CHKPURGE routine is called.

   a. The caller's protect key is saved, if necessary, in the SAMB, and is valid until the next EXCPVR SVC is issued.

   b. If this is an invalid EXCPVR SVC, and it was issued before a previous one completed, the CALLRTM macro is issued (with abend code 172).

   c. The flag, SAMPSTNX, is set off if a POINT or WRITE was issued.

   d. The flag, SAMSMF, is set on if the SMF EXCP count is to be updated.

   e. If a purged re-EXCPVR request must be re-created, CHKPURGE determines what, if any, modifications to the SAMB and the first IOB in the active IOB queue are required.

4. The IKNIT routine is called and initializes SAMB fields and flags, and builds a new channel program.

   a. For buffer DASD, builds prefix for buffered DASD.

   b. For nonbuffered DASD, builds standard prefix.

5. The CKVIORPS routine is called to point the IOB to the correct CCW for RPS and non-RPS DASD.

6. The SEGTEST routine is called to test if a segment process is beginning or ending. If so, SAMB fields are adjusted for this and control is returned to the pagefix appendage (PGFXAPP).

7. If an end-of-data condition has not been detected (flag SAMPSTNX is zero) or if the data set is open for update processing, the CPBLD routine is called to satisfy I/O requests on the ICQE.

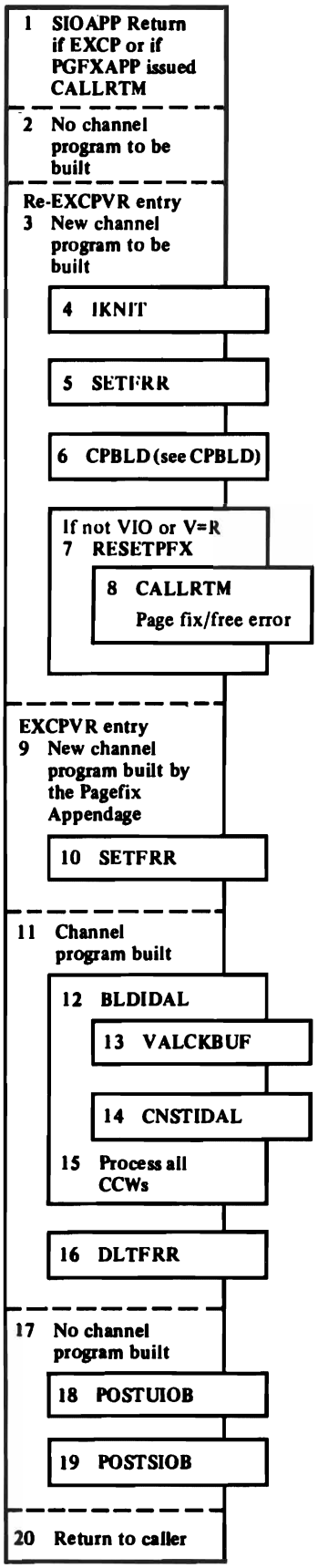
8. The address of the pagefix list and its size are determined and placed in registers 10 and 11.

   a. If the CPBLD routine was called, and the device is not a VIO device and the program is not running in a V=R region, the pagefix list address and length are obtained from the SAMB.

   b. Otherwise, there is no pagefix list, and registers 10 and 11 are zeroed.

9. EXCPVR returns control with the pagefix list address and length in registers 10 and 11.

SIO Appendage (SIOAPP): This compendium illustrates the SIO appendage.



The SIO appendage (SIOAPP) operates as follows:

1. If flag RQE114 is on, an EXCPVR SVC was issued. If flag RQE114 is off, an EXCP SVC was issued and is ignored by this appendage.

   If flag SAMCLRTM is on, the pagefix appendage issued the CALLRTM macro, and control is returned to EXCPVR using the return offset established by the CALLRTM routine. Otherwise, the return offset is set to zero, which is the normal return to EXCPVR. The pagefix list used for the previously executed channel program is saved. Otherwise, processing continues with Step 2 or Step 8.

2. If flag SAMSIO is on, a re-EXCPVR exit was taken from the channel end/abnormal end appendage, and the channel program in the SAMB is to be executed without modification by the SIO appendage. The process that rebuilds the channel program is skipped, and SIO appendage processing resumes with Step 20.

   If flag SAMPSTNX is on (an end-of-data condition was previously reached) and if the data set is not open for update processing, no channel program is built, and processing resumes with Step 17.

3. If flag SAMACT is on, this is a re-EXCPVR and a new channel program must be built.

4. The IKNIT routine is called. It initializes SAMB flags and fields for building a new channel program.

5. The SETFRR routine is called. An FRR routine is established prior to building a new pagefix list in CPBLD.

6. The SIO appendage calls the CPBLD routine (described below) to build a channel program and pagefix list.

7. If a VIO device is being accessed or if the buffers are in a V=R region, no pages are fixed and the pagefix processing is skipped; otherwise, the RESETPFX routine is called. The entries in the new and old pagefix list are compared. If each list has one entry and the entries are identical, processing continues. Otherwise, the saved pagefix list is pagefreed and the new pagefix list is pagefixed.

8. CALLRTM is called if any error is encountered page freeing of page fixing (ABEND 800). Control continues with Step 11.

9. If the SAMACT flag is not on, this is an EXCPVR entry and a new channel program has been built by the pagefix appendage. SAMACT is set on.

10. The SETFRR routine is called prior to building IDA words and validity checking buffers.

11. If no channel program was built (SAMSEGCT = 0), no IDA words are built; otherwise, BLDIDAL routine is called.

12. Each CCW with an IDA flag set to 1 is processed by BLDIDAL. If the caller's protect key is zero and if the caller's region is V=R, buffer validation is not performed.

13. Otherwise, the VALCKBUF routine is called and the SIO appendage checks the validity of the CCW's buffer address.

14. If a VIO device is being accessed or if the buffers are in a V=R region, IDA words are not built for the CCW and the CCW's IDA flag is set to zero. Otherwise, a list of IDA words is built in the next free slot in SAMIDAW. The address of the newly built IDA words is placed in the CCW by the CNSTIDAL routine.

15. If more CCWs in the channel program have the IDA flag on, each CCW with the IDA flag on is processed by BLDIDAL as described in Steps 12 through 15.

16. The FRR routine is deleted.

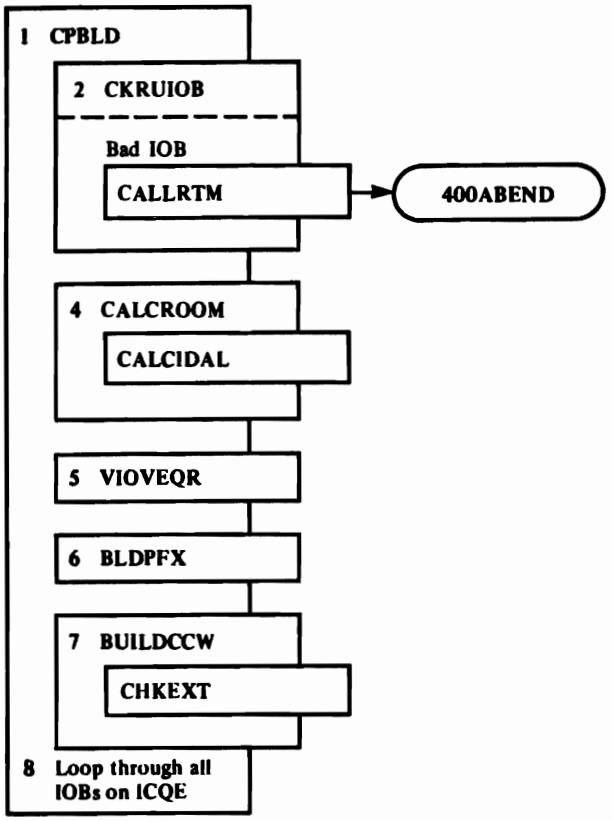
17. If no channel program was built (SAMSEGCT zero) or if an end-of-data condition has been encountered and the SAMB does not contain an update-only channel program, POSTUIOB is called.

18. POSTUIOB posts the first IOB on the active IOB queue (pointed to by the ICQE). The first IOB's ECB is posted with X'41'. If there is an end-of-data condition, the CSW indicates a "unit exception" condition; otherwise, the CSW indicates a "channel program check" condition. The return offset is set to the "ignore" return.

19. If the "ignore" return is to be taken (SAMRTNOF set to the ignore return offset), the POSTSIOB routine is called and the SAMB IOB is posted with X'41'.

20. The return offset (established in SAMRTNOF) is added to the return address. SIOAPP returns to EXCPVR.

The Channel Program Build Routine (CPBLD): This compendium illustrates the Channel Program Build routine.



The channel program build (CPBLD) routine operates as follows:

1. The CPBLD routine obtains the first IOB on the queue (pointed to by the ICQE).

2. The CKRUIOB routine is called. It checks the validity of the IOB. The IOB is checked to see that it is within the storage area obtained for the ICQE and IOBs during OPEN processing. If the IOB is not within the expected storage area, an ABEND with code 400 is scheduled by calling the CALLRTM routine.

3. CKRUIOB moves the IOB's data to the SAMB IOB; flag byte IOBNFLG1 and data in the IOBEX are moved from the user IOB to the SAMB IOB area.

4. The CALCROOM routine is called to see if a channel program for this IOB will be built. It inspects the type of request and the current status of the SAMB, and determines whether there is enough room to build IDA words for the buffer (in routine CALCIDAL). CALCROOM also checks to see if there is enough room for the CCWs needed. If the channel program for the request cannot be built, control is returned to the caller of CPBLD.

5. CALCROOM tests to see if a VIO device is being accessed or if the buffers are in a V=R region. If either condition is so, VIOVEOR is called to validity check buffers.

   a. It tests to see if the caller key is zero. If so, no restriction is placed on the buffer location.

   b. If a VIO device is being accessed, the buffer address may be anywhere except in the CSA. If the buffer address (in IOBCCW1) overlaps the CSA, the local lock does not ensure that pages remain fixed; the IOB's channel program is not built, and control is returned to the caller of CPBLD.

   c. If executing in a V=R region, and the device being accessed is not a VIO device, the buffer address must be within the V=R region boundary. If the buffer extends beyond the end of the V=R region, the IOB's channel program is not built, and control is returned to the caller of CPBLD.

6. If neither a VIO device nor a V=R region, BLDPFX is called. The pages, which include the buffer, are added to the pagefix list. If there isn't enough room in the pagefix list to add the buffer's pages, the IOB's channel program is not built, and control is returned to the caller of CPBLD.

7. The BUILDCCW routine is called.

   a. It builds the CCWs which satisfy the IOB's request, and adds the CCWs to the channel program.

   b. In the CHKEXT routine, if an invalid IOBSEEK address is in the user IOB, the CCWs won't be built and control is returned to the caller of CPBLD. An invalid IOBSEEK address can occur only for the first IOB on an active IOB queue.

8. If the IOB just processed is not the last IOB on the active IOB queue (pointed to by the ICQE), the next IOB on the queue is obtained. The CPBLD routine continues processing each IOB on the active IOB queue (commencing with 2 above) until no more IOBs remain to be processed. Control is then returned to the caller of CPBLD.

Channel-End Appendages

Channel-end appendages, if present, gain processor control when the I/O interruption supervisor reaches the channel-end appendage exit. For data sets, appendages distinguish between valid and invalid block lengths by computation.

The channel-end appendages are:

IGG019BT This appendage schedules the writing of successive blocks when a record has to be segmented.

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.

IGG019CU This appendage disconnects chained channel programs that have executed, posts their completion, and performs normal channel-end and abnormal-end appendage processing.

IGG019EI This appendage distinguishes between fixed, fixed-blocked, and undefined user blocks and embedded DOS checkpoint records. In the case of fixed-length blocked records, it also distinguishes between wrong-length and truncated blocks.

IGG019EJ This appendage distinguishes between wrong-length and variable-length blocks and embedded DOS checkpoint records.

IGG019BZ This appendage handles channel-end processing and abnormal-end processing for EXCPVR requests.

Appendage IGG019BT (Channel End—Create BDAM): Module IGG019BT schedules the writing of successive blocks when a record has to be segmented. The OPEN executor selects and loads this module if the DCB specifies:

Write (Load)

Variable-length spanned record

The module operates as follows for a channel-end condition:

- It receives control when the I/O supervisor arrives at the channel-end exit.
- It determines whether the WRITE was WRITE-SZ. If it was WRITE-SZ, the routine returns control to the I/O supervisor.
- When the WRITE-SF is issued, it determines whether the block was spanned record. If not, the routine returns control to the I/O supervisor.
- When a spanned record is being processed, the routine determines whether the entire record has been written. If the record has been written, the routine returns control to the I/O supervisor. When the entire record has not been written, the routine schedules the asynchronous exit routine. The asynchronous exit routine will schedule an EXCP to write a middle segment or the last segment of the record.

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

The channel-end appendage operates as follows:

- It tests to see if the EXCP was issued from an SVC routine. If so, it effects a normal return to EXCP.
- It performs length checking for fixed-length records. The SLI bit in the read-data CCW is left off. If a wrong length record is read, the command chaining bit is turned off and the CSW reflects channel end and wrong length indication. The channel-end appendage determines whether the record is a valid short block. For standard format-F records with a valid short block, the module turns on the EOV bit in the DCB and ECB.
- For nonstandard format-F records with the track-overflow feature, a short block is treated as a valid record.

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, or OUTIN

and the DCB specifies:

Variable-length records

(Under these conditions, the SLI flag is off in the read CCW.)

The module performs a length check for variable-length records.

The channel-end appendage operates as follows:

- It receives control when the I/O interruption supervisor arrives at the channel-end exit.
- It tests to see if the EXCP was issued from an SVC routine. If so, it effects a normal return to EXCP.
- 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 and the device is magnetic tape, a check is made to see if the block has been padded up to 18 bytes or block size, whichever is less. If so, the appendage turns off the error indicators in the ECB and DCB and returns control to the I/O supervisor. If the device is not magnetic tape or the block is not padded, 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 IGG019CU (Channel End, Abnormal End—Chained Channel-Program Scheduling): 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 chained channel-program scheduling end-of-block routines.) The OPEN executor selects and loads this appendage for use as the channel-end 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 channel-end and abnormal-end appendage exits.
- It tests to see if the EXCP was issued from an SVC routine. If so, it effects a normal return to EXCP.
- It tests to determine if the CSW and the "First ICB" field in the IOB, point to the same channel program.
- If they do, the appendage continues as it would for a channel-end condition.
- 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 it would for a channel-end interruption with a permanent buffer.

  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 disconnecting process until the IOB and the CSW point to the same channel program.

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

- It sets the IOB and the ICB to show that the channel program completed without an 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 to schedule these channel programs.
- If there are no more channel programs to be executed, the appendage returns control to the I/O supervisor.

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 it would for permanent errors.

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

- It isolates the channel program in error by disconnecting 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 takes a re-EXCP exit, which causes the error segment to be retranslated and retried.

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

- It receives control after the I/O supervisor error-retry procedure is found unsuccessful in correcting the error.
- For a 3211 Printer, it tests to see whether further retry is necessary. If the ECB is posted in error with an X'41' or X'44' and the command-retry bit in sense byte 1 is on, then it schedules the asynchronous-error-processing module, IGG019FS, and exits.
- It posts the ICB to show that the channel program was completed in error.
- It disconnects 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.

Appendage IGG019EI (Channel End, Abnormal End—Fixed-Length or Undefined-Length Record Format): Appendage IGG019EI distinguishes between fixed, fixed-blocked, and undefined user blocks and embedded DOS checkpoint records. In the case of fixed-length blocked records, it also distinguishes between wrong-length and truncated blocks. The OPEN executor selects and loads this appendage if the OPEN parameter list specifies:

Input, readback

and the DCB specifies:

OPTCD=H (specified in JCL)

Magnetic tape

Fixed, fixed-blocked, or undefined-length blocks

The appendage operates as follows:

- It receives control from the I/O interruption supervisor when the I/O interruption supervisor arrives at the channel-end and abnormal-end appendage exits.
- It tests to see if the EXCP was issued from an SVC routine. If so, it effects a normal return to EXCP.
- Upon encountering a checkpoint header record, bit 0 in the DEBTFLGS field of the DEB is turned on. It is turned off when the checkpoint trailer record is encountered. This provides the means to differentiate between the user's data records and the embedded checkpoint records.
- In the channel-end entry into this appendage, the number of bytes read is tested. If 20 bytes were not read and the bypass-flag bit in the DEBTFLGS field is off, the appendage takes the normal exit to the I/O interruption supervisor for fixed-length and undefined-length formats and performs the necessary record length check for fixed-block records. If 20 bytes were read, the record is tested to determine if it is a checkpoint header record. If it is not a checkpoint header record, the normal exit to the I/O interruption supervisor is taken for fixed-length and undefined-length formats, and record length checking for fixed-block formats is performed.
- When a checkpoint header record is encountered, the bypass-flag bit in the DEBTFLGS field is turned on, the DCBBLKCT field is decremented by the value in the IOBINCAM field, the "Flags 1-3" fields of the IOB are reinitialized, and the IOBERRCT field is set to zero. For QSAM, the IOB completion code is set to X'50' and the normal exit is taken to the I/O interruption supervisor. The bypassing of the checkpoint records is performed in the QSAM routines. For BSAM, the re-EXCP exit is taken to the I/O interruption supervisor.
- The appendage is reentered when the reexecuted channel program ends for BSAM or when the rescheduled channel program ends for QSAM and, finding the bypass flag on, tests for the checkpoint trailer record. If the record read is not the trailer record, the DCBBLKCT field is decreased, the IOB-flag fields reinitialized, and the IOBERRCT field is set to zero. For BSAM, the re-EXCP exit is taken to the I/O interruption supervisor. For QSAM, the IOB completion code is set to X'50', and the normal exit is taken to the I/O interruption supervisor. This process continues until the trailer record is read. When the trailer record is read, the bypass flag is turned off, and the above procedure is followed. The next entry to this channel-end appendage follows the reading of the record immediately following the embedded checkpoint records.
- The appendage is entered in the event of an abnormal condition arising. If this entry is the result of any condition other than a data error, control is returned to the I/O interruption supervisor by way of the normal exit.
- If it is a data error, a test is then performed to determine if a checkpoint header/trailer record was read. This test comprises an initial 12-byte comparison of the record's first 12 bytes with the checkpoint identifier

  /// CHKPT //

Should this comparison fail, a byte-by-byte comparison is performed. If 10 or more bytes compare successfully, it is then assumed that a header or trailer record has been encountered and the appendage returns control to the I/O interruption supervisor.

Appendage IGG019EJ (Channel End, Abnormal End—Variable-Length Record Format): Appendage IGG019EJ distinguishes between variable-length and wrong-length blocks and embedded DOS checkpoint records. The OPEN executor selects and loads this appendage if the OPEN parameter list specifies:

Input

and the DCB specifies:

OPTCD=H (via JCL)

Magnetic tape

Variable-length blocks

The appendage operates as follows:

- It tests to determine if the EXCP was issued from an SVC routine. If so, it effects a normal return to EXCP.
- It receives control from the I/O interruption supervisor when the I/O interruption supervisor arrives at the channel-end and abnormal-end appendage exits.
- Upon encountering a checkpoint header record, bit 0 in the DEBTFLGS field of the DEB is turned on. It is turned off when the checkpoint trailer record is encountered. This provides the means to differentiate between the user's data records and the embedded checkpoint records.
- In the channel-end entry into this appendage the first two bytes of the record are tested to determine if it is a valid block. (The first 2 bytes of a variable-length physical record specify the block length and are used in performing length-checking.) The first 12 bytes of a checkpoint header or trailer record (which are identical and 20 bytes in length) identify it as a header/trailer record. These 12 bytes are:

  /// CHKPT //

  The first 2 bytes of the checkpoint header record do not satisfy the length check as a variable-length record. If the first 2 bytes do satisfy the length check, the appendage takes the normal exit to the I/O interruption supervisor for variable-length records. If the first 2 bytes do not satisfy the length check for a variable-length record, the number of bytes read is computed. If 20 bytes were not read and the bypass-flag bit in the DEBTFLGS field is off, the appendage returns to the I/O interruption supervisor. If 20 bytes are read, the record is tested to determine if it is a checkpoint header record. If it is not a checkpoint header record, the normal exit to the I/O interruption supervisor is taken for variable-length formats.
- When a checkpoint header record is encountered, the bypass-flag bit in the DEBTFLGS field is turned on, the DCBBLKCT field is decreased by the value in the IOBINCAM field of the IOB, the "Flags 1-3" fields of the IOB reinitialized, and the IOBERRCT field set to zero. For QSAM, the IOB completion code is set to X'50' and the normal exit is taken to the I/O interruption supervisor. The bypassing of the checkpoint records is performed in the QSAM routines. For BSAM, the re-EXCP exit is taken to the I/O interruption supervisor.

- The appendage is reentered when the reexecuted channel program ends for BSAM or when the rescheduled channel program ends for QSAM and, finding the bypass flag on, tests for the checkpoint trailer record. If the record read is not the trailer record, the DCBBLKCT field is decreased, the IOB flag fields reinitialized, and the IOBERRCT field is set to zero. For BSAM, the re-EXCP exit is taken to the I/O interruption supervisor. For QSAM, the IOB completion code is set to X'50', and the normal exit is taken to the I/O interruption supervisor. This process continues until the trailer record is read. When the trailer record is read, the bypass-flag is turned off and the above procedure followed. The next entry to this channel-end appendage follows the reading of the record immediately following the embedded checkpoint records.

- The appendage is also entered in the event an abnormal condition arises. If this entry is the result of any condition other than a data error, control is returned to the I/O interruption supervisor by way of the normal exit.

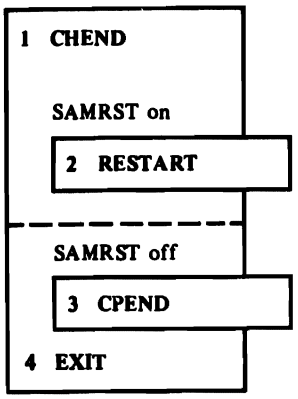
- If it is a data error, a test is then performed to determine if a checkpoint header/trailer record was read. This test comprises an initial 12-byte comparison of the record's first 12 bytes with the checkpoint identifier

  ```
  /// CHKPT //
  ```

  Should this comparison fail, a byte-by-byte comparison is performed. If 10 or more bytes compare successfully, it is then assumed that a header or trailer record has been encountered, and the appendage returns control to the I/O interruption supervisor.

Appendage IGG019BZ (DASD EXCPVR Channel End and Abnormal End): Appendage IGG019BZ does all channel-end and abnormal-end appendage processing for EXCPVR requests, for all record formats. The module includes two entry points: one for channel-end processing, and one for abnormal-end processing. Separate functions are performed at each entry point, and a common routine, CPEND, is called to process the channel program.

Channel-End Appendage (CHEND):
This compendium illustrates the channel-end appendage.



The channel-end appendage (CHEND) for EXCPVR requests operates as follows:

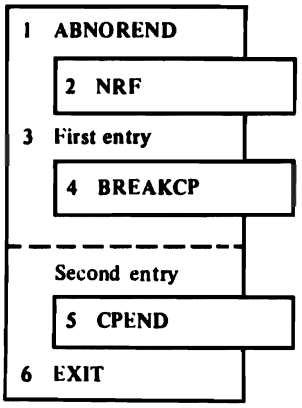
1. It is entered when a channel program completes. A test is made to see if the channel program resulted from an EXCPVR request. If not, the request is ignored and control is returned.

2. If a previously broken channel program is to be restarted (the SAMRST flag is on), it prepares the second half of the channel program (in the SAMB) for execution. The only channel program that can be broken is a PUTX request (update WRITE with refill READ). When the channel program is ready for execution, the return offset is set to 8 to re-EXCPVR.

3. For normal channel-end processing (SAMRST off), CHEND calls the CPEND routine to process the completed channel program segment(s) in the SAMB.

4. If the return offset is zero, the "active" flag (SAMACT in the SAMB) is set to zero. This allows an EXCPVR SVC entry to the pagefix appendage.

Abnormal-End Appendage (ABNOREND):
This compendium illustrates the abnormal-end appendage (ABNOREND) for EXCPVR requests.

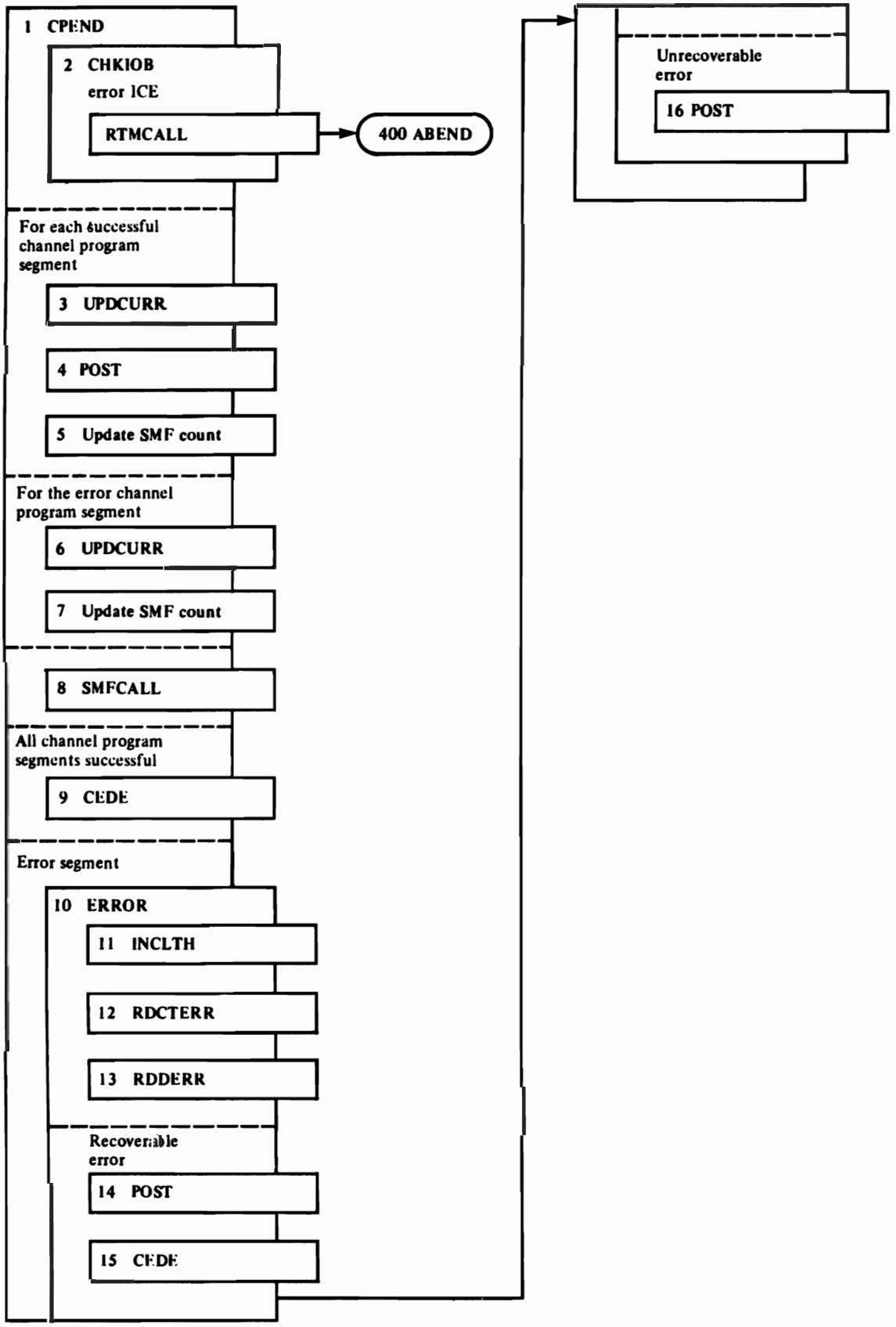


The abnormal-end appendage (ABNOREND) operates as follows:

1. ABNOREND verifies that the condition resulted from an EXCPVR request, and, if not, it ignores the request and returns control.
2. NRF is called. If the error is a "no-record-found" condition, and if it is expected (update or track overflow output), the seek address (in the SAMB) is updated and the TIC command (in the prolog CCW area) is reset to point to the failing channel program segment. The SAMSIO flag is set on to indicate that the channel program need not be rebuilt by the SIO appendage (SIOAPP).
3. If the error is not an expected no-record-found condition, and if the abnormal-end appendage is being entered for the first time, the BREAKCP routine is called.
4. If the request was QSAM PUTX (update WRITE with refill READ) and the error was in the update WRITE part of the channel program, the chained program is broken or separated so that only the update WRITE portion is subsequently executed by the ERPs. The SAMRST flag (in SAMB) is set on. If the error occurred in the refill READ portion of the channel program, only the READ portion is reexecuted by the ERPs. The return offset is left zero to allow ERP processing to occur.
5. If the appendage is being entered for the second time, the abnormal-end appendage calls the CPEND routine to process the channel program segments.
6. If the return offset is zero and this is the second entry, the "active" flag (SAMACT in the SAMB) is set to zero to allow an EXCPVR entry to the pagefix appendage.

This page intentionally left blank.

Channel Program Error (CPEND): This Compendium illustrates the CPEND routine of IGG019BZ appendage (EXCPVR requests).



The channel program end (CPEND) routine for EXCPVR requests operates as follows:

1. It is called from the channel-end appendage and the abnormal-end appendage to process completed channel program segments. The CPEND routine first processes all successfully completed channel program segments in the SAMB, then processes any segment that may have ended in error.

2. CHKIOB is called to validity check each IOB. If invalid, the RTMCALL routine issues a 400 ABEND.

3. UPDCURR is called, and the user IOB is updated to reflect the status of its related request.

4. POST is called, and the user IOB related to the completed channel program segment is posted as successful (X'7F'). The IOB is posted by the system post routine, IEAOPTB1. The address of the posted IOB is saved (in SAMIOBP, in the SAMB), and the address of the current IOB and the ICQE's pointer to the first IOB on the queue (ICQFIRST) is updated so that it points to the next IOB.

5. After the successfully completed user IOB is posted, the SMF count (in SAMSMFCT, in the SAMB) is incremented.

6. For abnormally completed channel program segments, UPDCURR is called and it is determined which CCWs were executed.

7. The SMF count is updated if the error occurred after a read data CCW.

8. When the IOBs of all successful channel program segments in the SAMB have been posted, the CPEND routine calls SMFCALL, if necessary, and IEASMFEX is called, passing the EXCP count (in SAMSMFCT, in the SAMB) to the SMF routine.

9. If all channel program segments completed successfully, the CEDE routine is called to determine if the last posted IOB is the last IOB on the active queue. If so, the return offset is set to cause a normal exit (return offset=0) to EXCPVR. If there is another IOB on the queue, the return offset is set to cause a re-EXCPVR exit (return offset=8).

10. The ERROR routine is called to check if the error is recoverable.

11. The INCLTH routine checks for incorrect-length. Incorrect-length indications on variable-length and undefined-length records formats are accepted, because the CCWs that are built do not have the SLI flag set on. An incorrect-length indication on a fixed-length standard record format is recognized as a software end-of-file indication. The "post next" flag (SAMPSTNX) is set on to indicate end of data on the next request.

12. The RDCTERR routine checks for a multitrack READ count error. When the IOB's EOB condition code is C'42', end of extent has occurred. End-of-extent processing was formerly handled by an end-of-extent appendage. If another extent is in the DEB, the seek address (IOBSEEK) is updated.

    The RDDERR routine checks for errors on READ DATA or READ KEY AND DATA commands. When the command is part of a "search previous" operation, the seek address is updated to skip the record in error and a re-EXCPVR return is indicated.

13. If the error is recoverable, POST is called (see Step 4).

14. For recoverable errors, CEDE is called (see Step 9).

15. When the error is not recoverable, POST is called and the request's IOB is posted with the unsuccessful completion code, X'41'. SAMRTNOF is set to cause a normal return to EXCPVR.

Program Controlled Interruption (PCI) Appendage (Execution of Channel Programs Scheduled by Chaining)

If chained channel-program scheduling is used in V=R, its address is placed into the appendage vector table for all three I/O interruption supervisor exits: PCI, channel-end, and abnormal end. In V=V, only the channel-end and abnormal-end entries are made.

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 the usual way after it regains processor 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 processor control at other I/O interruption supervisor appendage exits if chained channel-program scheduling is used.

Appendage IGG019V6 (PCI, Channel End, Abnormal End—Chained Channel-Program Scheduling): This appendage is the same as IGG019CU, described above, except that it is loaded into the V=R region instead of LPA and uses the PCIPOST macro to post ICBs because it does not hold the local lock. When it receives control for a PCI interrupt, it processes the ICBs that precede the one with the interrupt (as described above in IGG019CU). When the ICB that had the interrupt is found, the appendage returns to the I/O supervisor.

Abnormal-End Appendages

The abnormal-end appendage receives control from the I/O interruption supervisor when the latter finds a unit check condition in the channel status word (CSW). The appendage for this exit is a 3211 error appendage.

Appendage IGG019FR (Abnormal End—3211 Printer): Appendage IGG019FR schedules the asynchronous error-processing routine IGG019FS when a print line buffer (PLB) parity error or a UCS buffer parity error occurs.

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 are performed 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 inline 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 testing for the printer-overflow condition.

There are three CNTRL routines in QSAM; they are load modules. Figure 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.

Access Method Options	Selections		
CNTRL	X	X	X
Printer	X		
Card reader, single buffer		X	
3525 (printing)			X
Modules			
IGG019CA		CA	
IGG019CB	CB		
IGG019FA			FA

Figure 14. Module Selector—Control Modules

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

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

Figure 15. Control Routines That Are Expansions of Macro Instructions

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 that stacks the card just read, reads the next card into the buffer, forces an EOB condition to be recognized by the GET routine, and returns control to the processing program. (Card reader GET module IGG019AG depends on the use of this routine to refill empty buffers.)
- For BSAM, the module schedules a channel program that 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.
- If the 3505 or 3525 is specified, processing continues for stacker 1 or 2 (whichever is specified in the CNTRL macro instruction of the user's program).
- A test is made to determine if either OMR or RCE is being used.

  If either OMR or RCE is specified, the OMR/RCE bit is turned on in the operation codes of the CCWs.

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.

Control Module IGG019FA: This module performs line control functions if:

- The 3525 is specified
- A print file is specified
- CNTRL is specified

The module operates as follows:

- The line counter total (DCBLNP) in the DCB is increased according to the specifications in the CNTRL instruction.
- I/O macro sequencing is performed when using this module and a 3525 associated data set. If an error is detected, an abend (003) is issued with a return code of 03.
- If a skip to a channel on the next card is issued by the user, this module issues an EXCP to feed the next card,

issues a WAIT, and returns control to the user's program by way of register 14.

Printer-Overflow Macro Expansions: The PRTOV macro expansions permit processing program response to printer-overflow conditions.

The following macro expansions are created as inline 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.

Note: 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 the bit.

BASIC SEQUENTIAL ACCESS METHOD ROUTINES

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 seven types of BSAM routines:

- READ routines
- WRITE routines
- End-of-block routines
- CHECK routines
- Appendage routines
- Control routines
- SVC Routines

Diagram G, BSAM/BPAM Flow of Control, shows the relationship of the BSAM routines to other portions of the operating system and to the processing program.

Control routines (not shown in Diagram G) 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 instruction is encountered in a processing program. The track balance routine receives control from a WRITE routine or the track-overflow, end-of-block routine.

The BSAM control modules and routines are described later in this manual.

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.

Figure 16 on page 98 lists the modules 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 figure shows, for example, that module IGG019BH is selected and loaded if update and the READ macro instruction are specified.

Access Method Options	Selections								
Input	X			X					X
Output		X		X		X	X	X	X
INOUT, OUTIN	X	X							X
Update					X				
READ	X				X				X
Offset READ				X					
WRITE		X							X
WRITE (Load mode) (Create-BDAM)				X			X	X	X	
Fixed-length record format							X		X	X
Undefined-length record format								X		X
Variable-length record format				X	X			X		X
Spanned records				X						X
Track overflow									X	
*, DATA, or SYSOUT specified on DD statement										X
READ/WRITE Modules										

Figure 16 (Part 1 of 2). Module Selector—READ and WRITE Modules

Access Method Options	Selections
IGG019BA	BA BA
IGG019BH	BH
IGG019BR	BR
IGG019BU	BU
IGG019DA	DA
IGG019DB	DB
IGG019DD	DD
IGG019DK	DK

Figure 16 (Part 2 of 2). Module Selector—READ and WRITE Modules

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 to later test execution of the channel program.
- It fills in a CCW 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/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 WRITE macro instruction.
- If data is to be read into a buffer, the module flags the IOB for a READ operation, 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 and completes the length and buffer address entries in the WRITE CCW.
- The module passes control to end-of-block module IGG019TV. On return of control from that module, it returns control to the processing program.

WRITE Module IGG019BR (Create BDAM/VRE): Module IGG019BR writes variable-length spanned blocks and record-zero blocks for a data set that will later be processed by BDAM. The OPEN executor selects and loads this module if the DCB specifies:

WRITE (Load mode)

Variable-length spanned record

BFTEK=R

The module consists of three routines: one to write data blocks, one to write record-zero blocks, and an asynchronous exit routine.

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 from the EOV routine (to write the block not written into the previous volume) after the EOV routine of I/O Support has obtained another extent.
- It determines whether this block fits on the current track. If it does, the routine determines whether the new track balance is greater than 8 bytes. If the new track balance is equal to or less than 8 bytes, the routine adds write-capacity-record CCWs to the write-count-key-and-data CCWs. It then issues an EXCP.
- If the block does not fit on the current track, the routine determines whether the block fits on the current volume. If it does, this module constructs a channel program to write the first segment from a segment area associated with this IOB and to write the capacity record of this track. It then issues an EXCP. The asynchronous exit routine writes the successive segments. The DCBFDAD field has the address of the highest track on which the last segment of this record is written.
- If the block does not fit on the track or within the current volume, this routine constructs a channel program to write the capacity record of the track. It then issues an EXCP.

The asynchronous exit routine writes the capacity records of all the tracks on this volume. The EOV routine reschedules the WRITE request on the same volume spanning the extents, if the secondary allocation is on the same volume. When the secondary allocation is on a different volume, the WRITE request is written on the new volume.

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 EOV 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 EOV 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.

The asynchronous exit routine operates as follows:

- It receives control from the channel-end appendage through the exit effector when a spanned record is to be processed.
- If the record is a spanned record, it constructs a segment from the remaining part of the record and issues an EXCP macro instruction to write the segment.
- If the record is a spanned volume record, it issues an EXCP macro instruction to write capacity records up to the end of the extent.

READ Module IGG019BU: This module completes the channel program to read a direct data set, and relates the control blocks used by the I/O Supervisor to the channel program. The OPEN executor selects and loads this module along with an associated channel-end appendage (IGG019BV) if the OPEN parameter list specifies:

Input

and the DCB specifies:

BFTEK=R

Variable-length spanned record format for a BDAM data set with keys

The module operates as follows:

- It receives control when a READ macro instruction is encountered in the processing program.
- It enters the address of the IOB into the DECB to permit the CHECK routine to later test execution of the channel program.
- It waits for the SAMB IOB ECB (in the ICQE) to complete, then tests it to determine if the count field of the record to be read is available.
- If the count field of the record to be read is not available (as indicated by the SAMSCHPR flag in the SAMB), an EXCPVR macro instruction is issued. If the SAMSCHPR flag is set and BFTEK=R processing has been requested, only the count field of the next record will be read.
- When the count field of the record to be read is available, the buffer address and record length are placed in the CCW.

- The module then issues an EXCPVR macro instruction.

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

WRITE (Load mode)

Fixed-length record format

With the rotational position sensing (RPS) feature, this module tests the first CCW of a channel program created by IGG0199L. It tests for a set-sector command to determine whether it should take any RPS CCWs into account when making modifications to the channel program.

The module operates as follows:

- It receives control from the processing program when it encounters a WRITE macro instruction and also from the EOV routine after the end-of-volume routine of O/C/EOV 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 routine from which it received control.
- If this is the last block of a track (that is, no other block fits 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 variable-length and undefined-length blocks and record-zero blocks for a data set to be processed later by BDAM. The OPEN executor select and loads this module if the DCB specifies:

WRITE (Load mode)

Variable-length or undefined-length record format

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

With the rotational position sensing (RPS) feature, the module tests for a set-sector command in the first CCW of a channel program created by IGG0199L. If it is an RPS channel program,

the module makes the necessary modifications to the channel program.

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 from the EOV routine (to write the block not written into the previous volume) after the end-of-volume routine of O/C/EOV 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 routine from which it received control:

  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 updates 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 mode)

Fixed-length record format

Track overflow

With the rotational position sensing (RPS) feature, the first CCW of a channel program created by IGG0191M is tested by this module for a set-sector command code. If the code is present, alterations to the channel program are made accordingly.

The module operates as follows:

- It receives control from the processing program when the program issues 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 it would 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.

READ/WRITE Module IGG019DK: (SYSIN/SYSOUT): Module IGG019DK interfaces with a job entry subsystem to obtain records from the system input stream or to pass records to the system output stream for a BSAM processing program. The OPEN executor selects and loads this module if the open parameter list specifies:

Input, output, INOUT, OUTIN (*, DATA, or SYSOUT coded in the DD statement)

and the DCB specifies:

READ, WRITE

Fixed, undefined, or variable-length records

The module consists of READ and WRITE routines and a CHECK routine (SYSOUT only). The SAM module, IGG017BB, processes the CHECK macro instruction for SYSIN (see Figure 17 on page 106). (See Diagram M for an overview of SAM-SI processing for BSAM.)

The READ routine operates as follows:

- It receives control after a READ macro instruction is issued in the processing program.
- The RPL is initialized and the DCB is examined to determine if blocked records are specified. If they are, the number of I/O operations specified in the I/O counter field (CIIOCNT) is determined by the DCB block size and record length. If the records are not blocked, the I/O counter field is set to 1. The format of the CICB is shown in JES3 Data Areas microfiche.
- A JES GET request is issued. The request is issued as many times as is necessary to satisfy the count in the I/O counter field. The return code passed by the job entry subsystem in register 15 is checked by the READ routine.
- If an end-of-data condition is detected, the DECSDECB is posted with X'42' and control is returned to the processing program. The DECSDECB is posted with X'41' for a permanent error.
- If the return code indicates a successful completion, control is returned to the processing program with the DECSDECB posted with X'7F'.

The WRITE routine operates as follows:

- It receives control after a WRITE macro instruction is issued in the processing program.
- The RPL is initialized and the number of I/O operations required to process the WRITE macro instruction is determined. The number is placed in the I/O counter field (CIIOCNT) of the CICB.
- If the DCB record format indicates ASA or machine control characters, then the control character is checked to determine if it is a Composed Page Data Stream control byte. In this case, the ACB data stream indicator is set (ACBCCDSI) before passing control to the job entry subsystem (JES).
- A JES PUT request is issued. The request is issued as many times as necessary to satisfy the count in the I/O counter field.
- When processing is completed, control is returned to the processing program. The ECB is set to X'7F' for a normal completion and a X'41' for an error.

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. This occurs 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.

If the American National Standard Code for Information Interchange (ASCII) is used and input is specified, the check module issues an XLATE macro instruction which translates the entire input buffer from ASCII form to EBCDIC form. If format-D records are specified, the record descriptor words are first converted from decimal to binary. For format-D records when BUFOFF ≠ L, the length of the record read is calculated and placed in the DCB LRECL field.

Figure 17 on page 106 lists the available CHECK routines and the conditions that cause a particular module to be used. The OPEN executor selects one of the six routines, loads it, and places its address into the DCBCHECK field. For example, Figure 17 shows that module IGG019DC is selected and loaded if a WRITE for a BDAM create is specified.

Access Method Options	Selections					
Input					X	
* or DATA specified on DD statement					X	
Output	X		X			X
SYSOUT specified on DD statement						X
INOUT, OUTIN	X					
Update		X				
WRITE	X					
WRITE (Load) (Create-BDAM)			X	X		
Variable-length spanned record format			X			
CHECK Modules						
IGG019BB	BB				BB	
IGG019BI		BI				
IGG019BS			BS			

Figure 17 (Part 1 of 2). Modules Selector—CHECK Modules

Access Method Options	Selections
IGG019DC	DC
IGG019DK[1]	DK

Figure 17 (Part 2 of 2). Modules Selector—CHECK Modules

Note to Figure 17:

[1] The CHECK routine described in this section is part of the BSAM processing module IGG019DK listed in Figure 16 on page 98.

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. If ASCII coding is used, the entire input buffer is translated from ASCII to EBCDIC. If ISO/ANSI/FIPS spanned record format (DS/DBS) is used, the 5-byte ISO/ANSI/FIPS segment control word (SCW) is converted to the IBM 4-byte segment descriptor word (SDW), which leaves an unused byte at the beginning of each segment. If a SYSIN data set is being processed and an error condition is detected, control is passed to the SAM-SI SYNAD routine, IGG019AH.

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.
- A test is made to determine if a SYSIN data set is being processed.
- If SYSIN was not specified, processing continues in the normal manner.
- If SYSIN was specified, the completion code in the ECB is tested.
  - If a completion code of X'7F' was returned, control is returned to the processing program.
  - If a completion code of X'42' was returned, indicating an end-of-data condition, the EOV SVC (55) is issued. For concatenated data sets, control is returned to this routine. If DCBOFLGS specifies "unlike" attributes, control is returned to the calling program immediately. Otherwise, the read routine indicated in DCBREAD is entered to reschedule the request. Control is returned to the user upon completion of CHECK processing for this request.
  - If any other completion code was returned, module IGG019AH is loaded and entered with a BALR instruction. This is the error-processing module for SYSIN/SYSOUT. (See Figure 12 on page 67). The error-processing module is deleted if control is returned to the CHECK module. (User SYNAD routine may not return control.)

- 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. Two types of returns from the EOV routine are possible:
  - If the EOV routine determines the error condition to be an EOV condition, the routine passes control to the end-of-volume routine of O/C/EOV for volume switching. That routine passes control to the EOV/new volume routine, which reschedules the purged channel programs; this routine then returns control to the CHECK module.
  - If the EOV routine determines the error condition to be a permanent error, the routine 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 the CHECK routine, the routine issues a second SVC 55 instruction. The routine 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 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 processes 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 has not yet completed processing, the module issues 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.
- On return of control from the EOV routine, the CHECK module passes control to the processing program's SYNAD routine. If the SYNAD routine returns control to the CHECK routine,

the routine issues a second SVC 55 instruction. The routine 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 IGG019BS (Create BDAM): Module IGG019BS 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 mode)

Variable-length spanned record

BFTEK=R

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 a user specifies WRITE-SFR, the next record address (TTR) is supplied in the next address field of the DECB.
- If the execution of the channel program encounters 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 a DMABCOND macro instruction to ABEND.
- If the WRITE routine encounters an EOV condition and therefore does not request scheduling of the channel program for execution, this module issues an SVC 55 instruction. On return of control, this module tests for completion of the channel 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 mode)

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 a DMABCOND macro instruction to ABEND.
- If the WRITE routine encountered an EOV condition and therefore did not request scheduling of the channel program for execution, this module issues an SVC 55 instruction. On return of control, this module tests for completion of the channel program.

CHECK Module IGG019DK (SYSOUT): The CHECK routine in this module receives control after a CHECK macro instruction is issued in the processing program for a SYSOUT data set. (See Diagram M for an overview of JES compatibility interface processing for BSAM.)

The CHECK routine operates as follows:

- The return code in the DECSDECB is tested.
  - If a completion code of X'7F' is found, control is passed back to the processing program.
  - If a completion code of X'41' is found, indicating an I/O error, module IGG019AH (error-processing module for SYSIN/SYSOUT; see Figure 12 on page 67) is loaded and entered with a BALR instruction. The error-processing module is deleted if control is returned to the CHECK routine.
- Control is returned to the processing 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 I/O devices.

There are three types of BSAM control routines:

- Routines that are loaded into processing program virtual storage by the OPEN executor (CNTRL, NOTE/POINT).
- Routines that are loaded into supervisory transient area by an SVC instruction in a processing program macro expansion or in another control routine, such as BSP or track balance. See "SVC Routines" on page 154.
- Routines that are inline 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.

Figure 18 on page 111 and Figure 19 on page 111 list the various kinds of control routines and the conditions that cause them to gain control. Figure 18 shows the access condition options that cause the OPEN executor to load a control routine for use with a DCB.

Figure 19 lists the different macro expansions constructed by the assembler.

Access Method Options	Selections						
Note/Point	X	X	X				
Chained scheduling			X				
CNTRL		X			X	X	X
Direct-access storage			X				
Magnetic tape	X	X		X			
Card reader					X		
Printer						X	
3525 (printing)							X
Control Modules							
IGG019BD	BD						
IGG019BE		BE					
IGG019BK			BK				
IGG019BL				BL			
IGG019CA[1]					CA		
IGG019CB[1]						CB	
IGG019FA[1]							FA

Figure 18. Module Selector—Control Modules Selected and Loaded by the Open Executor

Note to Figure 18:

[1] These routines are also used in QSAM; see Figure 14 on page 95 for a description of these routines.

Macro Instructions	Number of Macro Expansions
PRTOV—User exit	1
PRTOV—No user exit	1

Figure 19. Control Routines that Are Expansions of Macro Instructions

Notes to Figure 19:

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

This figure duplicates Figure 15 on page 95; it is repeated here to identify all control routines available in BSAM.

Control Module IGG019BD (NOTE/POINT—Magnetic Tape): 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 the NOTE macro instruction 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 tests for the bypassing embedded DOS checkpoint records option by testing bit 3 of the DCBOPTCD field. If the option is found to have been specified, the routine issues a GETMAIN to obtain 20 bytes and modifies the CCW to read the first 20 bytes of each block into the obtained virtual storage while performing recording positioning. The suppress-incorrect-length-indication bit is set in the CCW. The actual bypassing of any embedded DOS checkpoint records is performed by either channel-end appendage IGG019EI or IGG019EJ. Module IGG019BD uses the FREEMAIN macro instruction to obtain virtual storage prior to returning to the user.
- 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 tapemark, 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 results 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 Tapemark, 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 tapemark (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 tapemark 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 tapemark.
- 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 tests bit 3 of the DCBOPTCD field for the bypassing embedded DOS checkpoint records option. If the option is found to have been specified, the routine issues a GETMAIN to obtain 20 bytes and modifies the CCW to read the first 20 bytes of each block into the obtained virtual storage while performing record positioning. The suppress-incorrect-length indication bit is set in the CCW. The actual bypassing of any embedded DOS checkpoint records is performed by either channel-end appendage IGG019EI or IGG019EJ. Module IGG019BD uses the FREEMAIN macro instruction to obtain virtual storage prior to returning to the user.
- 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 tapemark 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 direct-access processing. The OPEN executor selects and loads this module if the DCB specifies:

POINT

Direct-access storage

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 is READ and more than one buffer is used, the channel program last executed places the FDAD value into the IOBSEEK field in the IOB.
- If the macro instruction is READ and only a single buffer is used, the channel program last executed places the FDAD value into the DCBFDAD field of the DCB.
- If the macro instruction is WRITE, the end-of-block routine places 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.

If the records are fixed-length standard format, the record number is passed to the resident sector routine to compute the sector value. If the record format is not fixed-length standard format, the value 0 is placed in the byte used by the set-sector CCW.

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 which 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 decreased by 1; if Z is 1, the address calculated by the convert routine is left unchanged. For an explanation of how the value of Z is set, see the description of the POINT macro instruction in Data Administration: Macro Instruction Reference.

- It inserts the value into the DCBFDAD and IOBSEEK fields if track overflow or update is being used. It sets the DCBOFLGS field to show that the contents of the DCBTRBAL field are no longer valid.
- It determines if the new full disk address references a new device address (that is, a new UCB). If a new device is addressed, SVC 25 is issued to update the IOBSEEK field in the SAMB's IOB. Control then returns 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. 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. If a tapemark, load point, or permanent error is encountered in the execution of one of the channel programs, the routine sets the DCBOFLGS field to indicate a permanent error. (Subsequent attempts by the READ or WRITE routine to cause scheduling of channel programs for execution results in their not being scheduled. On the next entry into the CHECK routine, the condition is detected and handled.)

Control Module IGX00030 (MSGDISP—Magnetic Tape—Display message to tape drive): This module receives control through an SVC 109 when the MSGDISP macro is issued.

This module builds an eight character message partially supplied by the caller and displays it on the screen for the tape drive, either through its own IOS driver (when the caller wants to wait for I/O completion) or through the MISCELLANEOUS IOS driver (when the caller does not want to wait for I/O completion). It provides several ways of displaying different types of messages corresponding to the type of request made by the caller through the MSGDISP macro as follows.

READY: It displays a 6 character message supplied by the caller.

VERIFY: It displays a 6 character volume serial number and one character label type to signify that the volume is accepted.

MOUNT: It displays the character "M" together with the volume serial and label type to request that the operator mount the volume.

DEMOUNT: It displays a disposition character "K", "R", or "D" for keep, retain, and demount respectively, to inform the operator of the disposition of the currently mounted volume.

RESET: It clears all messages in the display area and displays the internal state of the hardware.

GEN: It displays one or two eight character message(s), depending on whether one or two message(s) are supplied by the caller.

Control Module IGX00031 (SYNCDEV—Magnetic Tape—Synchronization of Buffered Write Data): This module gets control through an SVC 109 when the SYNCDEV macro is issued.

This module uses the NOTE ABS service to determine the buffering depth of the device. It then does the following, depending on the type of service requested:

INQUIRY ROUTINE: It returns the buffering depth to the caller.

SYNCHRONIZATION ROUTINE: If the buffering depth exceeds the limit specified by the caller, it builds and executes a channel program to force synchronization on the device. Otherwise, it returns the buffering depth.

Control Module IGX00032 (NOTE/POINT—Magnetic Tape—Physical Block ID): This module gets control when a NOTE or POINT macro is issued with the ABS parameter specified.

This module provides two functions:

NOTE ROUTINE It builds and executes a channel program to record the physical block IDs of the record that is about to enter or leave the buffer.

POINT ROUTINE: It builds and executes a channel program to position the tape to the record whose physical block ID is supplied by the caller. (This physical block ID may be obtained from a previous NOTE.)

BASIC PARTITIONED ACCESS METHOD (BPAM) ROUTINES

A partitioned data set has a directory and members. The directory is read and written using BPAM routines, whereas the members are read and written using BSAM routines. (See 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; for 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 store and retrieve entries in the directory and convert direct-access addresses from relative to absolute. 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 conversion routines refer to the data extent block (DEB) to determine the address value complementary to the given value.

BPAM routines (see Figure 20 on page 117) 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 virtual storage. Figure 20 on page 117 shows how these routines gain control.

See "SVC Routines" for descriptions of BPAM routines.

BPAM Routines	Module Number	Residence	Instruction Passing Control
STOW	IGC0002A	Link pack area	SVC 21
STOW	IGG0210A	Link pack area	XCTL from IGC0002A
STOW	IGG021AB	Link pack area	XCTL from IGG0210A
FIND (C option)	(Macro Expansion)	User program	FIND (C option)
FIND (D option)	IGC018	Nucleus	SVC 18
BLDL	IGC018	Nucleus	SVC 18
Convert TTR	IGC018	Nucleus	BAL IECPCNVT
Convert MBBCCHRR	IGC018	Nucleus	BAL IECPRLTV
Convert sector	IGC018	Nucleus	BAL IEC0SCR1

Figure 20. BPAM Routines Residence

DUMMY DATA SET

Dummy Data Set Module IGG019AV: 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.

SEQUENTIAL ACCESS METHOD EXECUTORS

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, see Open/Close/EOV Logic. Figure 21 on page 117 indicates the other figures that describe the OPEN and CLOSE executors. Executors perform processing unique to an access method when a data control block is being opened or closed.

OPEN executors

CLOSE executors

Executor	Number	Receives Control From	Via	Passes Control To
OPEN	See Figures 22, 23, and 24	See Diagram E	XCTL (WTG Table)	See Diagram E

Figure 21 (Part 1 of 2). Sequential Access Method Executors—Control Sequence

Executor	Number	Receives Control From	Via	Passes Control To
CLOSE	See Figure 25	Close routine	XCTL (WTG Table)	Close routine See Figure 25

Figure 21 (Part 2 of 2). Sequential Access Method Executors—Control Sequence

The executors reside in the link pack area. It is the OPEN executors that load 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. It constructs the data extent block (DEB), the buffer pool if requested, input/output blocks (IOB), the channel programs, and, if chained channel-program scheduling is used, interruption control blocks (ICB). It selects and loads 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 virtual storage used by the IOBs, ICBs, and channel programs.

DCB RELOCATION TO PROTECTED WORK AREA

Before control is passed to SAM OPEN executors, the DCB is copied to the OPEN/CLOSE/EOV work area to ensure the integrity of DCB vectors that could be changed by the user during system open time or system close time. The DCB copy is updated by SAM executors during open processing and is used to refresh the user's DCB prior to the initiation of any I/O operation. (The user's DCB is used for all I/O initiated during open, except in the validation modules, which use the DCB copy.) All I/O is completed and the SAM work area, IOBs, and the DEB are updated to reflect the location of the user's DCB within the user's address space before control is returned to common open. SAM executors refresh the user's DCB from the work area copy.

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 the last load of the OPEN executors. The stage 2 executors in turn, pass control to the stage 3 executors. 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. To pass control to the next executor that is to process the data set, each executor issues an IECRES macro with the XCTL and BRANCH=DIRECT parameters. This macro generates a branch to the OPEN/CLOSE/EOV service routine, IFG019RA, which branches to the next executor. For a description of the WTG table, see Open/Close/EOV Logic.

When an ABEND is to be issued by an OPEN or CLOSE executor, it issues a DMABCOND macro to prepare to pass control to IGG0196M or IGG0206M. (These two problem determination modules are described in Open/Close/EOV Logic.) A DMABCOND macro is issued instead of an ABEND macro because the problem determination routines write a message to the user, issue the GTRACE macro to trace pertinent control blocks, and call the optional DCB ABEND exit routine before possibly issuing an ABEND macro.

System modules that build, delete, or modify data extent blocks (DEBs), use DEB validity checking, a separate routine that protects the user's data from unauthorized access. The modules must maintain a table of DEB pointers in protected storage by use of the DEBCHK macro instruction, described in Data Administration. The logic of the DEBCHK routine is in Open/Close/EOV Logic.

The OPEN executors maintain an audit trail in the OPEN work area to indicate which resources have been acquired. This audit trail is interrogated by the force CLOSE executor when a force close situation arises.

The message text for all messages issued by the OPEN executors are contained in the message CSECT. Before issuing a message, the executor must extract the message text from the message CSECT.

Diagram E shows the flow of control among the three stages of OPEN executors.

Stage 1 OPEN Executors

Stage 1 OPEN executors construct data extent blocks (DEBs), build buffer pools, and issue DEBCHK (TYPE=ADD) macros. If a printer with the universal character set (UCS) feature and/or a forms control buffer (FCB) is specified, the executors call SETPRT to perform printer setup.

If UCS/FCB images are not specified in the user's JCL, the executors must ensure that the current images, as specified by the UCB UCS extension, are default images. If none are loaded or the loaded images are not default images, the operator must specify which images are to be used.

The OPEN routines determine which executor is required to begin processing of each DCB specified in the OPEN parameter list. For SAM processing, the entry placed in the WTG table is IGG0191A for an actual data set, IGG0191C for a dummy data set, and IGG0199F for a SYSIN or SYSOUT data set.

The executor for the first entry in the WTG table gets control from the common OPEN routines.

As each stage 1 executor completes its processing, the name of the next executor (for the DCB being processed) is placed in the WTG table. Then a check is made to determine, for each entry in the OPEN parameter list, if another DCB requires the use of the executor now in control. If so, the executor is reentered as many times as necessary to process all of the entries in the WTG table requiring this executor. If no other DCBs require this executor, control is passed to the next executor that is specified in the WTG table (starting from the top of the list) for a DCB that has not completed its processing. For a particular DCB, all of the stage 1 executors are executed before control is passed to a stage 2 executor.

Figure 22 on page 119 lists the access method conditions that cause different stage 1 executors to be selected, loaded, and to receive control after loading. The executors are described in the text that follows. The order of presentation is the same as that shown in Figure 22 under Executors.

In Figure 22, an X in a column represents a condition that must be satisfied for the executor marked in that column. A blank in the upper portion of the table indicates that either the condition is not required for selection or not examined at this time. The table should be used in conjunction with the flow of control information in Diagram E, SAM Flow of Control for OPEN executors.

Access Method Options	Selections	Selec-tions	Selec-tions	Selec-tions	Selec-tions
Actual data set	X X X	X X X	X X	X X X	X X
Dummy data set			X		
*, DATA, or SYSOUT in DD statement					X
3505 (OMR/RCE) or 3525				X	
Direct-access device	X	X		X X	
Printer with UCS Feature (1403, 3203, 3211, 3262 Model 5, 3800, 4245, 4248)		X			
Printer with forms control buffer (3203, 3211, 3262 Model 5, 3800 4245, 4248)		X			
Buffer pool required	X	X		X X	
User totaling specified			X X	X X	
Executors					
IGG0191A	1A 1A 1A	1A 1A 1A	1A 1A	1A 1A 1A	1A
IGG0191B	1B 1B 1B	1B 1B 1B	1B 1B	1B 1B 1B	1B
IGG0191C			1C		
IGG0191I	1I	1I		1I 1I	1I
IGG0191N	1N	1N		1N 1N	
IGG0191Y			1Y 1Y	1Y 1Y	
IGG0193I	3I 3I 3I	3I 3I 3I	3I 3I	3I 3I 3I	3I
IGG0196A	6A 6A 6A	6A 6A 6A	6A 6A	6A 6A 6A	6A
IGG0196B	6B 6B 6B	6B 6B 6B	6B 6B	6B 6B 6B	6B
IGG0196I	6I 6I 6I	6I 6I 6I	6I 6I	6I 6I 6I	6I
IGG0196Q		6Q 6Q			
IGG0197L				7L	
IGG0197M				7M	
IGG0197U		7U			
IGG0199F					9F
IGG0199G					9G
IGG0199W					9W

Figure 22. OPEN Executor Selector—Stage 1

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 tests the OPEN macro option against the DCBMACRF field. It issues an 013 ABEND via a DMABCOND macro if any of the conditions listed are found. The conditions are:

  For QSAM:

  that buffer length is not smaller than block size if the buffer length is specified

  that the block size is not at least 4 bytes larger than logical-record length for variable-length records

  that logical-record length (if specified) is not equal to block size for fixed-length unblocked data sets

  For BSAM and QSAM:

  that block size is not an even multiple of logical record length for fixed-length blocked data sets

- It performs a test to determine if the block size is an integral multiple of the logical record length (LRECL) for QSAM with fixed blocked records or BSAM data sets. If the block size is not an integral multiple of LRECL, QSAM data sets are abnormally terminated with an abend (013).

- If search-direct has been requested (OPTCD=Z in the DCB and a direct-access device is being used), the executor determines if search-direct can be supported (that is, not VS, UT, FBS, etc.) and sets the bit in JFCBMASK+6 to X'08' if the request can be honored. For a 3890 MICR device, it will issue a DMABCOND macro instruction if RECFM, BLKSI, LRECL, or BUFL are specified incorrectly.

- The executor specifies in the WTG table that module IGG0196I is the next module required for this DCB.

- It searches the WTG table to pass control to another executor.

Stage 1 OPEN Executor IGG0191B: Executor IGG0191B is loaded after executor IGG0196A, IGG0191N or IGG0191Y has completed processing all entries in the WTG table.

The executor operates as follows:

- It stores DCBLRECL in the DEB.

- It sets DCBCNTRL to 0.

- If the device type is direct-access storage, the address of the device table is stored in the DCB. From the device table, the key overhead (or 0 if there are no keys) and track balance are stored in the DCB.

- If the JFCB indicates a partitioned data set, the DSCB and the DSORG field of the DCB are checked to be sure they specify partitioned organization. If not, a DMABCOND macro instruction is issued.

- If partitioned organization is specified:

  - For direct-access OUTPUT or OUTIN, the track balance of the last write, from the DSCB, is stored in the DCBTRBAL.

- For direct-access input, the member name from the JFCB is stored in the DEB. The routine then issues a BLDL macro instruction to find the extent. If BLDL returns a nonzero return code, a DMABCOND macro instruction is issued.

The executor issues a BALR to the convert routine at CVTPCNVT to convert the TTR to MBBCCHHR and stores it in DCBFDAD.

- If the data set is not partitioned, DCBFDAD is set to DEBBINUM. If a dummy extent is indicated, the DCBFDAD + 3 is set to X'FF' to indicate an illegal FDAD.
- If unit record equipment is specified, for input only and NOTE/POINT is requested, DCBCNTRL + 1 is set with ID of the dummy routine.
- If LRECL is not specified, DCBLRECL is initialized for QSAM.
- The executor specifies in the WTG table that module IGG0196B is the next module required for this DCB.
- It then searches the WTG table to pass control to another executor.

Stage 1 OPEN Executor IGG0191C: Executor IGG0191C operates as follows:

- It receives control from the OPEN routine if the DD statement is DUMMY.
- It issues a GETMAIN macro instruction for a DEB.
- It does a DEBCHK, ADD, to put the DEB in the DEB table.
- It issues a LOAD macro instruction for IGG019AV and stores the characters 'AV' in the subroutine ID section of the DEB.
- It issues a GETMAIN macro instruction for buffer space and constructs the buffers.
- It sets various audit trail bits.
- It issues a DMABCOND macro instruction if BUFLEN and BLKSIZE are not specified for QSAM.
- The executor specifies in the WTG table that IGG0191I is the next executor for this DCB.

Stage 1 OPEN Executor IGG0191I: Executor IGG0191I is loaded after IGG0196B, unless the OPEN executors must load UCSB or FCB images. In this case, it is loaded after IGG0197M.

The executor operates as follows:

- If a buffer pool has already been built, the executor gets virtual storage for a record area for QSAM logical record interface.
- If the values in both the DCBBUFL and DCBBLKSI fields are zero, the executor issues a DMABCOND macro instruction.
- If time sharing (TS) is specified with BSAM and DCBBUFL and DCBBLKSI fields are zero, it sets the length of the buffer to terminal line length. When QSAM is specified and DCBBUFL and DCBBLKSI fields are zero, it sets the length of the buffer to logic record length. If DCBLRECL field is also zero, it sets the length of the buffer to terminal line length.
- 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 DCBBUFNO field determines the number of buffers constructed.

- If the DCB specifies blocked records, a unit record device, output, and not undefined RECFM, it turns off the blocked-records bit in DCBRECFM. For fixed-length records, it sets DCBBLKSI equal to DCBLRECL.

- If logical record interface is required for variable-length spanned records processed in locate mode, the executor adds a length of 32 bytes plus the maximum logical-record length, which is specified in the DCBLRECL field for a record area, to the size of virtual storage required. If extended logical record interface (XLRI) is specified, the DCB LRECL value is a multiple of 1024, which is used to calculate the size of the record area, and the 32-byte control field is added. Eight more bytes (including 4 bytes of padding) are added to the buffer control block to store the address of the record area. Flags are set (X'C0') to indicate an extended buffer control block and the presence of the record area.

- It issues a DMABCOND macro instruction if BUTEK=A is specified and teh processing mode is not locate.

- It stores the length of the entire record area in the first word of the record area.

- It specifies that executor IGG0193I is required for this DCB in the WTG table. It then searches the WTG table to pass control to another executor.

Stage 1 OPEN Executor IGG0191N: Executor IGG0191N receives control after executor IGG0196A. It supplements executor IGG0196A by building the device-dependent portion of the DEB for direct-access devices.

For partitioned data sets, it sets the authorized library table bit in the DEB if the data set is in the authorized library table.

If the data set resides on an MSS virtual volume, an ICBACREL macro is issued to allocate space on and/or stage data to a direct-access device. If MSS window processing has been requested by the user, a flag is set on in the DEBXFLG1 field in the DEB extension, providing the MSS data set is (1) physical sequential in organization, (2) allocated in cylinders, and (3) being processed by QSAM or BSAM for INPUT or OUTPUT only (not INOUT, OUTIN, nor UPDAT). Any errors result in abnormal termination (see abend code 413, return code 2C, in the "Diagnostic Aids" section of this manual). If a partitioned data set resides on an MSS virtual volume and will be opened for INPUT processing, the following options exist at the time the data set is opened:

- To stage the entire data set to end-of-file, specify OPTCD=H as a DCB subparameter on the associated DD statement.

- To stage only the directory of the data set, do not specify OPTCD on the associated DD statement.

Note: The OPTCD option may only be specified on the DD statement; it cannot be specified with the DCB macro.

The user label extent is not inserted into the DEB. This executor specifies either IGG0191B or IGG0191Y as the next entry in the WTG table for processing the DCB, unless the DCB specifies EXCP, in which case IGG01911 is the next executor for this DCB.

It then searches the WTG table to pass control to another executor.

Stage 1 OPEN Executor IGG0191Y: Executor IGG0191Y receives control after executor IGG0196A or executor IGG0191N when the user-totaling option has been specified in the DCB, that is, when bit 6 of DCBOPTCD is 1.

This executor operates as follows:

- It sets bit 7 of DCBOFLGS to 0 to prevent a successful open and issues a DMABCOND macro to write a message to the programmer for any of the following reasons:
  - No DCB exit list.
  - No totaling entry in DCB exit list.
  - Image area address is zero.
- It calculates the size of the area required to save the user's totaling areas and issues a GETMAIN to obtain the space.
- It constructs control blocks for the work area and places the address of the save area in the access method portion of the DEB. (Figure 34 on page 241 describes the access method save area.)
- It loads the resident save routine IGG019AX and places the ID of the save routine in the DEB and the address in the user-totaling save area.
- It specifies in the WTG table that executor IGG0191B is the next executor required. It then searches the WTG table to determine the next executor to receive control.

Stage 1 OPEN Executor IGG0193I: This executor receives control from IGG0191I.

The executor specifies which stage 2 executor is specified in the WTG table. The module selector table for stage 2 executors, Figure 25 on page 141, should be used to determine which stage 2 executor is required for this DCB.

For direct-access processing, if write load (that is, BDAM create processing) is not specified, IGG0193I passes control to IGG0193B. If write load is specified, IGG0193I passes control to IGG0191L.

For other than direct-access processing, if chained scheduling can be supported and has been requested (with OPTCD=C), an appropriate chained scheduling executor is specified in the WTG table.

If chained scheduling can be supported but has not been requested, tests are made to see if it can be given anyway without interfering with a dependence that the issuer of OPEN may have on normal scheduling. There are two cases in which OPEN cannot supply chained scheduling unless requested by the keyword OPTCD=C.

1. Printer—The PRTOV macro may be used and it does not operate properly with chained scheduling.
2. Reading format-U records—With chained scheduling, the actual length of the record is not available.

It then searches the WTG table to determine which executor receives control. The IECRES macro instruction is used to pass control to the next executor.

Stage 1 OPEN Executor IGG0196A: Executor IGG0196A receives control from and supplements IGG0196I.

- The executor issues a DEBCHK (TYPE=ADD) macro to add the newly created DEB address to a protected area table of DEB addresses.
- It completes the DEB construction initiated in OPEN executor IGG0196I.
- If the device type is a printer with a printer device characteristics table (PDCT), the DCBDEVT field contents are obtained from the PDCT.

  If the device type is not a printer with a printer device characteristics table (PDCT), the executor fills in the DCBDEVT field with the device type and number from the UCB. If unit record equipment is indicated, the UR bit in the DCBDEVT field is set.
- The executor specifies in the WTG table which module is the next one required for this DCB, as follows:
  - For direct-access—executor IGG0191N.
  - If the device type is a printer with UCS feature or is a 3800 Printing Subsystem and EXCP is specified—executor IGG0196Q.
  - If the device type is other than a printer and EXCP is specified—executor IGG01911, the final module of the OPEN executors.
  - If the device type is tape, not input, not EXCP, and the user-totaling facility is specified—executor IGG0191Y.
  - If the device type is other than a printer with the UCS feature or direct access, BSAM or QSAM is specified, and the user-totaling facility is not specified—executor IGG0191B.
- It then searches the WTG table to pass control to another executor.

Stage 1 OPEN Executor IGG0196B: Executor IGG0196B receives control from and supplements IGG0191B.

The executor operates as follows:

- For QSAM, DCBBUFNO is set to 5 (3 for 2540) if not previously specified.
- Executor issues DMABCOND macro instruction (calls problem determination module) if the buffer length is less than block size or if data set is for a printer and something other than output (only) is specified.
- Determines the next executor to receive control.
  - For a time sharing (TS) task, control is transferred to IGG0196S unless buffers are wanted. If buffers are needed, the OPEN routine transfers control to IGG0191I.
  - A test is made to determine if either the 3505 (without OMR or RCE) or 3525 is being used, just prior to the XCTL subroutine. If either device is being used, control is passed to module IGG0197L; otherwise, normal processing continues.
  - If the device type is a printer with the UCS feature, IGG0196Q receives control.
  - If a buffer pool is required, IGG0191I receives control.

- Otherwise, IGG0193I receives control to select the stage 2 executor.

Stage 1 OPEN Executor IGG0196I: Executor IGG0196I receives control from and supplements IGG0191A.

The executor operates as follows:

- It computes the virtual-storage requirement for the DEB and obtains the space. The space does not include the user label extent, because it is reflected in the first extent field of a format-1 DSCB for a physical sequential or direct data set. If no primary extent has been requested for an output data set, as shown by the contents of the DS1NOEPV field of the DSCB, the executor sets the DCBCIND1 field to show a volume-full condition.
- It specifies in the WTG table that executor IGG0196A is the next executor required for this DCB.
- It then searches the WTG table to pass control to another executor.

Stage 1 OPEN Executor IGG0196Q: This executor initializes a printer with the UCS feature or a 3800 Printing Subsystem Models 1 and 3. It receives control from IGG0196A (EXCP) and IGG0196B (BSAM/QSAM).

The executor operates as follows:

- For printers other than the 1403, it obtains storage for the ERP work area if one does not exist. A use count is kept of the work area. Each time the module requests storage for the work area, the count is incremented by 1. Each time the work area is released, the use count is decremented by 1. When the use count is zero, the work area is freed by IGG0202J.
- It obtains storage for a SETPRT parameter list. If the JFCB indicates that a JFCBE (JFCB extension for the 3800 Printing Subsystem) exists, the SETPRT parameter list is then completed using the information in the JFCBE and the JFCB. If the JFCBE does not exist, the SETPRT list contains zeros in the device-dependent fields. If the device is a 3800 Printing Subsystem, the module turns on the SETPRT initialization bit, which allows the SETPRT executors to restore the 3800 Printing Subsystem to its hardware defaults before the device was set up with data set dependent requirements.
- If the device is a 3800 Printing Subsystem and SETPRT is not successful, message IEC162I with SETPRT return codes is issued, followed by an abend (IEC141I 013-CC).

  If the device is not a 3800 Printing Subsystem and SETPRT fails, the SETPRT return code is converted to the appropriate internal abend code for an IEC152I B13-CC abend.
- This executor then indicates in the WTG table the stage 2 executor to receive control for processing the DCB. The module selector table for stage 2 executors, Figure 25 on page 141, should be used.

Stage 1 OPEN Executor IGG0197L: Executor IGG0197L receives control from IGG0196B whenever the 3505 or 3525 is specified.

The executor operates as follows:

- It initiates registers with the addresses of the DCB, UCB, ECB, and CVT.
- A test is made to determine if either OMR or RCE is being used.

- If OMR is specified, a test is made to determine if the device is a 3525. If the device is a 3525, control is transferred to IGG0197M.
- If either OMR or RCE is specified, the format descriptor record is loaded and decoded.
- After the read-only has been executed and the format card has been translated, an OMR or RCE CCW is constructed and executed (writes the format of the device).
- It specifies in the WTG table that IGG0197M is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 1 OPEN Executor IGG0197M: IGG0197M receives control from IGG0197L.

The executor operates as follows:

- If an OMR or RCE format card is invalid, or if an invalid device is specified for OMR, this module issues a WTP message and an abend (004) with a return code of 05.
- If no invalid condition exists, the executor specifies in the WTG table the next module required for this DCB, as follows:
  - IGG0191I if QSAM is specified and no buffer pool control block exists.
  - IGG0197N if either BSAM or QSAM is specified and the user has specified a buffer-pool control block.
  - IGG0191I if BSAM is specified and the user has specified a buffer number but not a buffer buffer-pool control block.
- It then searches the WTG table to pass control to another executor.

Stage 1 OPEN Executor IGG0199F (SYSIN/SYSOUT): Executor IGG0199F receives control when the OPEN routines (see Diagram K) determine that the SAM-SI executors are required to process a DCB for a SYSIN or SYSOUT data set (*, DATA, or SYSOUT coded in the DD statement).

Note: If a device is directly allocated to the task and the OPEN verification subsystem interface (SSI) determines that the data set is to be handled by a subsystem, data management uses the SAM-SI executors and the SSI interfaces to access the device. The interface to the device subsystem is the same as the JES interfaces for a SYSIN/SYSOUT data set.

The executor operates as follows:

- It issues a GETMAIN macro instruction to obtain virtual storage for a JES compatibility interface control block (CICB). The format of the CICB is described in JES3 Data Areas microfiche.
- It constructs an ACB and an RPL in the CICB, for communicating with the JES, and initializes an SVC exit list with entries for BSP and SYNADAF SVCs.
- It supplies defaults to appropriate DCB fields in the open copy of the DCB.
- It specifies in the WTG table that executor IGG0199G is the next executor required for this DCB.
- It then searches the WTG table to pass control to another executor.

Stage 1 OPEN Executor IGG0199G (SYSIN/SYSOUT): Executor IGG0199G receives control from the SAM-SI OPEN executor IGG0199F.

The executor operates as follows:

- The WTG table is scanned and an open list is constructed to open an ACB for each SYSIN/SYSOUT entry in the WTG table.
- It issues an OPEN (type J) macro instruction for the ACBs just constructed.
- It chains the DEB, created by OPEN for the ACB, to the DCB. The address of the DCB is placed in DEBECBAD, leaving DEBDCBAD pointing to the ACB (see Figure 36 on page 244).
- It checks the DCB for invalid combinations of access method options. An abend (013) is requested (using problem determination routines) if any invalid combinations are found.
- It specifies in the WTG table that IGG0199W is the next executor required for this DCB.
- It then searches the WTG table to pass control to another executor.

Stage 1 OPEN Executor IGG0199W (SYSIN/SYSOUT): Executor IGG0199W receives control from the SAM-SI OPEN executor IGG0199G.

The executor operates as follows:

- It determines the buffer requirements, then obtains and chains buffer (if necessary).
- The RPL, contained in the CICB, is initialized according to the record format specified in the DCB.
- It issues a GETMAIN macro instruction to obtain a work area for collecting segments, if necessary.
- It specifies in the WTG table that IGG0198L is the next 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. Figure 23 on page 129 lists the access conditions that cause the different stage 2 executors to be loaded and to receive control.

For direct-access operations other than write load, IGG0193B serves as a stage 2 and 3 OPEN executor. IGG0193B returns control directly to IGG0190S, the I/O services common routine. IGG0193B builds all control blocks and loads all routines needed.

The device-oriented processing performed by a stage 2 executor primarily consists of the construction of input/output blocks (IOB), 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, executors also construct interruption control blocks (ICB).

Figure 23 lists the access conditions that cause the different stage 2 executors to be loaded and to receive control. The executors are described in the text that follows and are in the same sequence as the list in the figure.

Access Method Options	Selections	Selections	Selections	Selections	Selections
Track overflow		X			
Direct-access storage		X X			X
INOUT, OUTIN	X				
Unit record or magnetic tape or paper tape	X X X	X	X	X X X	
Write-Load (Create-BDAM)		X X			
Chained scheduling	X X				
3505			X	X	
3525				X X	
3890					X
OMR or				X	
RCE or				X X	
Print only and associated files				X	
TS terminal			X		
Open Executors[1]					
IGG0191G	1G		1G	1G	
IGG0191L		1L 1L			
IGG0191M		1M			
IGG0191Q	1Q				
IGG0191R	1R				
IGG0193B					3B
IGG0196K		6K			
IGG0196S[2]			6S		
IGG0197N			7N	7N 7N 7N	
IGG0197P				7P 7P	
IGG0197Q				7Q 7Q	
IGG0197V					7V
IGG0199L		9L			

Figure 23. OPEN Executor Selector—Stage 2

Notes to Figure 23:

[1] If *, DATA, or SYSOUT is specified on the DD statement, no stage 2 executors are loaded.

[2] See ACF/TCAM Diagnosis Guide and ACF/TCAM Diagnosis Reference.

In this figure, an X in a column represents a condition that must be met for the executor to be selected. A blank in the upper portion of the table indicates that either the condition is not required for selection or is not examined at this time. The table should be used in conjunction with the flow of control information in Diagram E, SAM Flow of Control for OPEN Executors.

Stage 2 OPEN Executor IGG0191G: Executor IGG0191G normally receives control from executors IGG0193I, IGG0196Q (for 3800 Printing Subsystem only), IGG0197F, IGG0197U, or IGG0197N. Under abnormal conditions, it receives control from executors IGG0191R, IGG0191Q (chained scheduling not supported) if:

- The DCB specifies BSAM or QSAM and either unit record or magnetic tape.
- The OPEN macro parameter is INOUT or OUTIN and the DCB specifies magnetic tape.

The executor operates as follows:

- It computes the amount of virtual storage required for the IOBs, issues a GETMAIN macro instruction from subpool 0, in the user's key, and then sets the virtual storage for the IOBs to zeros. It stores the number of bytes gotten for the IOBs in the second word of the audit trail for force close.
- When control is returned from GETMAIN, it sets an audit trail bit to indicate to the force CLOSE executor that storage should be freed.
- It then tests to see if the device type for this data set is unit record. If so, IGG0196K is specified in the WTG table for this DCB and the check for other DCBs that need this executor is made.
- If the device is not unit record, processing continues in this module. It constructs the channel programs in the IOBs and fills in the other fields of the IOBs. It stores the address of the first IOB in the DCB and sets the first IOB bit in the first IOB. If there is only one IOB for this data set, it sets the IOB unrelated flag.

The executor specifies in the WTG table the next executor required for this DCB. If the DCB specifies variable-length record format, the next executor is IGG01915. 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 IGG0191L: Executor IGG0191L receives control after executor IGG0193I if the DCB specifies:

Create-BDAM (Write-Load)

The executor constructs IOBs and enters the address of the first IOB into the DCB. Then it loads the create-BDAM WRITE, CHECK, and channel end appendages and inserts their addresses into the DCB.

It loads the create-BDAM channel end appendage and places its address in the DEB appendage vector table (AVT).

With the rotational position sensing (RPS) feature, more virtual storage is needed for the channel programs. This executor computes the extra bytes needed for the RPS channel programs and issues a GETMAIN. The sector bytes are placed at the end of all the IOBs and channel programs. The last doubleword of the GETMAIN area is used for sector manipulation. The first byte is used by set-sector and by read sector. The second byte is used

as a byte of zero on which to issue a set-sector command in order to position at the beginning of the track.

If track overflow is specified, the routine specifies that executor IGG0191M is the next executor required for this DCB. Otherwise, the routine specifies IGG0199L as the next executor required. It then searches the WTG table to pass control to another executor.

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

With the rotational position sensing (RPS) feature, more virtual storage is needed for the channel programs. This executor computes the extra bytes needed for the RPS channel programs and issues a GETMAIN. The sector bytes are placed at the end of all the IOBs and channel programs. The last doubleword of the GETMAIN area is used for sector manipulation. The first byte is used by set-sector and by read sector. The second byte is used as a byte of zero on which to issue a set-sector command in order to position at the beginning of the track.

The executor operates as follows:

- If the extents are smaller than the blocks, it issues a DMABCOND macro instruction to abend.
- 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 IGG0191Q: Executor IGG0191Q gains control after executor IGG0196Q (for 3800 Printing Subsystem only), IGG0197U, IGG0197F, or IGG0193I if the DCB specifies:

Chained channel-program scheduling

Unit record and magnetic tape

The executor operates as follows:

- 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.
- From subpool 0 in the user's key, 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 stores the number of

bytes gotten for the IOBs in the second word of the audit trail for force close.

- When control is returned from GETMAIN, it sets an audit trail bit to indicate to the force CLOSE executor that storage should be freed.
- It sets the PCI flag in the READ Count CCW, only if in a real address environment.
- For QSAM data sets with fixed-blocked record format on a unit record device, and the buffer pool was not gotten by OPEN, it sets DCBBLKSI equal to DCBLRECL and turns off the blocked records bit in DCBRECFM.
- If chained scheduling cannot be supported because of conflicting specifications, bit 5 of DCBCIND2 is set to 0 to indicate that chained scheduling is not being supported. The executor specifies IGG0191G as the next executor to receive control; otherwise, unless variable spanned record format is specified, bit 5 of DCBCIND2 is set to 1 to indicate support of chained scheduling and IGG01913 is specified as the next executor to receive control. When variable spanned or ISO/ANSI/FIPS variable spanned record format is specified, IGG01916 is the next executor 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 executor IGG0193I if the OPEN parameter list specifies:

INOUT, or OUTIN

and the DCB specifies:

Chained channel-program scheduling

Magnetic tape

The executor operates as follows:

- 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.
- From subpool 0 in the user's key, 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 stores the number of bytes gotten for the IOBs in the second word of the audit trail for force close.
- When control is returned from GETMAIN, it sets an audit trail bit to indicate to the force CLOSE executor that storage should be freed.
- It sets the PCI flag in the read count CCW, only if in a real address environment.
- If chained scheduling cannot be supported because of conflicting specifications, bit 5 of DCBCIND2 is set to 0 to indicate that chained scheduling is not being supported, the executor specifies IGG0191G as the next executor to receive control; otherwise, bit 5 of DCBCIND2 is set to 1 to indicate support of chained scheduling and IGG01913 is specified as the next executor to receive control.
- It then searches the WTG table to pass control to another executor.

Stage 2 OPEN Executor IGG0196K: Executor IGG0196K receives control if executor IGG0191G determines that the device type is unit record.

- This executor builds channel programs, using the storage gotten in IGG0191G.
- For QSAM fixed blocked record format, it sets DCBBLKSI equal to DCBLRECL and turns off the blocked records bit in DCBRECFM.

The executor specifies in the WTG table the next executor required for this DCB. If the DCB specifies variable-length record format, the next executor is IGG01915. 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 IGG0197N: Executor IGG0197N receives control from IGG0193I whenever the 3505 or 3525 is specified, or from IGG0197M whenever the same devices are specified and a buffer pool is not needed.

The executor operates as follows:

- It makes a test to determine if the FUNC parameter is being used.
- If the FUNC parameter is not being used, and if the file is for read-only (without OMR or RCE) or punch-only, IGG0191G is specified in the WTG table as the next executor required for this DCB.
- If the FUNC parameter specifies print only or associated files, IGG0197P is specified in the WTG table as the next executor required for this DCB.
- If a specified parameter combination is found to be invalid, a message to the programmer (WTP) is issued along with a subsequent abend (004).
- If the FUNC parameter is not being used, but the file is a read-only with OMR or RCE, IGG0197P is specified in the WTG table as the next executor required for this DCB.
- After the validity of the FUNC parameter is established, the DCBMACRF field is tested to determine if the CNTRL is valid for an input data set. If it is not valid, a WTP message and an ABEND macro (004) with a return code of 02 are issued.
- If the CNTRL specification is valid, a test is made to determine if the associated DCBs specify the same access methods.
- If the access methods are not the same, a message is written to the programmer along with a subsequent ABEND (004).
- It specifies in the WTG table that IGG0197P or IGG0191G is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 OPEN Executor IGG0197P: IGG0197P receives control from IGG0197N if neither read-only (without OMR or RCE) nor punch-only is specified for the 3505 or 3525.

The executor operates as follows:

- It builds the IOB and CCWs and appends a work area to the IOB, according to the type of data set that is specified.

- It specifies in the WTG table that IGG0197Q is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 OPEN Executor IGG0197Q: IGG0197Q receives control from IGG0197P.

The executor operates as follows:

- A test is made to determine if data protection image (DPI) is specified in the FUNC parameter.
- If DPI is specified, SVC 105 is issued. This builds a DCB for SYS1.IMAGELIB and returns its address in register one.
- Both a BLDL and a LOAD macro are issued so that the DPI image can be built and the image address can be loaded in register zero.
- The address is saved for the image deletion (after the image has been copied into IOB + 64) by the DELETE macro.
- If DPI is not specified, tests are made to determine which EOB and/or control module ID is to be entered in the DCB. (The same tests are made if DPI is specified.)
- It specifies in the WTG table that 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 IGG0197V: IBM 3890 Document Processor executor, IGG0197V, receives control after either executor IGG0196B or IGG0193I. For information about the executor, see OS/VS Logic for IBM 3890 Document Processor.

Stage 2 OPEN Executor IGG0199L: Executor IGG0199L receives control after executor IGG0191L if the DCB specifies:

Create-BDAM (Write-Load)

The executor constructs channel programs. When the DCB specifies RECFM=VS and BFTEK=R, the routine constructs a segment work area for spanned record processing and creates an IRB for the asynchronous exit routine, which executes writing of the successive segments. 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.

With the rotational position sensing (RPS) feature, more virtual storage is needed for the record-ready channel programs. This executor computes the extra bytes needed for the record-ready channel programs and issues a GETMAIN. The sector bytes are placed at the end of all the IOBs and channel programs. The last doubleword of the GETMAIN area is used for sector manipulation. The first byte is used by set-sector and by read-sector. The second byte is used as a byte of zero on which to issue a set-sector command in order to position at the beginning of the track.

Note: A user may provide a segment work area by setting a bit in the DCBMACRF field and placing the address of that area in the DCBEOB field. In this case, this routine will not construct the segment work area.

Stage 3 executors load 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.

Some of the modules to be loaded are identified by stage 2 executors having set codes in DCBCNTRL for unit record and tape data sets. The 4 bytes of DCBCNTRL identify these types of modules:

Byte	Module Type
+0	EOB (QSAM) or EOB for read (BSAM)
+1	EOB for write (BSAM)
+2	NOTE/POINT or CNTRL
+3	NOTE/POINT or CNTRL

Note that the first byte is DCBEROPT and is saved at DXCCW6 during stage 1 and restored by IGG01911. The codes that can be in the 4 bytes and the modules they can identify, depending on which stage 3 executor does the loading, are:

00	No module to load	
01	Reserved	
02	IGG019CC, IGG019CW	End-of-block
03	IGG019CE, IGG019CX	End-of-block
04	IGG019CF, IGG019CY	End-of-block
05	Reserved	
06	IGG019BD, IGG019BL	NOTE/POINT, tape
07	IGG019CA	CNTRL, card reader
08	IGG019CB, IGG019CC	CNTRL, printer or End-of-block
09	IGG019BE	CNTRL, tape
0A	IGG019AV	DUMMY or no-op for various functions
0B	IGG019CT	End-of-block, error
0C	IGG019TC	End-of-block, user-totaling
0D	IGG019TC	End-of-block, user-totaling
0E	IGG019TW	End-of-block, user-totaling
0F	IGG019TW	End-of-block, user-totaling
10	IGG019CT	End-of-block, error

In many of the above pairs, the first one is for normal scheduling and the second one is for chained scheduling.

The stage 3 OPEN executors load in the fixed standard end of extent modules and the format-U channel end module when the rotational position sensing (RPS) feature is used.

Figure 24 on page 136 lists the access conditions that cause the different stage 3 executors to be loaded and to gain control.

The executors are described in the text that follows in a sequence identical to the list under "Executors" in Figure 24.

In this figure, an X in a column represents a condition that must be satisfied before the executor is selected. A blank in the upper portion of the table indicates that either the condition is not required for selection or not examined at this time. The table should be used in conjunction with the flow of control information in Diagram E, SAM Flow of Control for OPEN Executors.

Stage 3 OPEN Executor IGG01910: IGG01910 receives control after executor IGG0197Q. It also receives control after executor IGG0191G.

This executor operates as follows:

- For QSAM it identifies, loads, and puts the address into the DCB of:

  A GET or PUT routine

  A synchronizing routine

- It loads appendages and puts their addresses in the DEBAVT.
- It puts end-of-block routine addresses in the DCB.
- If BSAM is specified, it identifies, loads, and places the addresses in the DCB of the READ/WRITE routine and the CHECK routine.
- For 3211 printers, it issues a CIRB macro instruction to create an IRB for an error retry module; it loads an abnormal end appendage and an error retry module.
- For user-totaling, it loads the EOB routine and places its address in the DCB.
- It enters into the DEBSUBID field of the DEB the identification of each routine loaded.
- It specifies executor IGG01911 in the WTG table as the next executor to receive control for this DCB.

Access Method Operations										
Chained scheduling	X	X								
None of the above					X		X			
*, DATA, or SYSOUT specified on the DD statement								X		
Variable-length records			X				X			
Dummy data set								X		
OPEN Executors:										
IGG01910					10					
IGG01911	11	11			11	11	11			
IGG01913	13									

Figure 24 (Part 1 of 2). OPEN Executor Selector—Stage 3

Access Method Operations				
IGG01915				15
IGG01916			16	
IGG0198L				8L

Figure 24 (Part 2 of 2). OPEN Executor Selector—Stage 3

Stage 3 OPEN Executor IGG01911: Executor IGG01911 is entered from executors IGG0191C, for dummy data sets; IGG0191N, IGG0196A, for EXCP data sets; and IGG01910, IGG01912, IGG01923, IGG01915, and IGG01916 for all SAM unit record and tape data sets.

This executor operates as follows:

- It issues the IECRES macro instruction to cause the user's copy of the DCB to be updated to reflect the changes and additions made by the OPEN executors to the protected copy of the DCB.
- It issues a DELETE macro instruction for the message CSECT if it was loaded by stage 1 OPEN executors.
- It sets an audit trail bit for the SAM/PAM/DAM force CLOSE executor to indicate the data set can be closed by the normal close executor string during force close processing.
- It puts the buffer address into the CCW and, if ANSI with BUFFOF=L is not specified, zeros the first four bytes (buffer chain pointer) of each buffer.

It sets the type of related request (RR) for normal scheduling data sets with more than 1 IOB. The type of RR setting defines the time in the processing of the current request that IOS can start the next RR. This is controlled by the processing REQD to prepare the next RR for I/O initiation. (IE, if chan end appendage updates the next IOB after completion of the current IOB, I/O cannot be started for the next request until IOS has received control back from the chan end appendage.) The following matrix shows the types of related requests.

Processing Required	Type I	Type II	Type III
DRP processing		X	
CE appendage		X	
CE interrupt			X
SIO appendage	X		
EOE appendage	X		

Note: The only exception to the above matrix is for mag tape. Unit exception is presented from trying to write over the reflective marker on reading a tape mark. To keep IOS from starting the next RR it is necessary for the ERP to mark this in permanent error. Since ERP processing is after CE processing (and SAM output data sets, SAM tape input data sets do not have ce appendages,) the type must be II.

- For data sets other than QSAM, it returns to common open.
- It completes any remaining DCB fields.
- It completes the IOBs.
- It puts the buffer address in the READ or WRITE CCWs for unit record and magnetic tape data sets. If an invalid buffer address is found, it issues a DMABCOND macro instruction.
- For QSAM input:

  Chained Scheduling: It chains all channel programs for move, data, and substitute modes. For locate mode, it chains together all but one. It then issues an EXCP macro instruction against the main IOB to prime the buffers.

  Normal Scheduling: It issues a GETMAIN macro instruction from subpool 230 in the user's key for a register save area for the access method routines. It saves the address returned from GETMAIN in the second word of the audit trail for force close. It then passes control to the EOB routine (BALR if the key is less than 8, SYNCH if the key is greater than 7) to prime the user's buffers (for all but one IOB if locate mode; all buffers for other processing modes). Before exiting, it frees the register save area.
- For output, it sets a flag, which is used to identify the first entry, into the PUT routine.
- 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 IGG01913: Executor IGG01913 receives control after executors IGG0191Q and IGG0191R if the DCB specifies:

Chained channel-program scheduling

The executor operates as follows:

- For 3211 printers, it issues a CIRB macro instruction to create an IRB for an error retry module; it loads an abnormal-end appendage and an error retry module.
- If QSAM is specified, it identifies, loads, and places the address into the DCB of:

  A GET or a PUT routine

  A synchronizing routine
- If BSAM is specified, it identifies, loads, and places the address into the DCB of:

  A READ or WRITE routine

  A CHECK routine
- It loads appendages and puts their addresses in the DEBAVT.
- It loads the end-of-block routines specified by the stage 2 executors.
- It specifies in the WTG table that OPEN executor IGG01911 is to receive control next for this DCB.

Stage 3 OPEN Executor IGG01915: Executor IGG01915 receives control after executors IGG0191G, IGG0196K, and IGG0197Q, if the DCB specifies:

Variable-length record format

The executor operates as follows:

- If QSAM is specified, the executor identifies and loads a GET or PUT routine and a synchronizing routine.
- If BSAM is specified, the executor identifies and loads a READ or WRITE routine, a CHECK routine, and a routine to service the NOTE/POINT macro instruction if it is specified.
- It loads appendages and puts their addresses in the DEBAVT.
- It loads the end-of-block routines specified by the stage 2 executors.
- It issues a DMABCOND macro instruction if LRECL=X is specified and the processing mode is not Locate.
- It issues a DMABCOND (abend code 013-4) macro to cause an abend if ISO/ANSI/FIPS records are to be processed and there is no record area present.
- It places the identifiers (IDs) of the routine loaded into the DEB subroutine ID field and the addresses of the routines into the DCB.
- For a 3211 printer:

  An abnormal-end appendage is loaded and its address is placed in the appendage vector table.

  An asynchronous error routine is loaded. The IRB used for scheduling this routine is built and the IRB address placed in the DEB.
- It specifies in the WTG table that executor IGG01911 is the next executor required for this DCB.
- It searches the WTG table to determine to which executor it should pass control.

Stage 3 OPEN Executor IGG01916: Executor IGG01916 receives control after executors IGG0191Q and IGG0191R if the DCB specifies:

Variable-length record format

The executor operates as follows:

If QSAM is specified, the executor identifies and loads a GET or PUT routine and a synchronizing routine.

If BSAM is specified, the executor identifies and loads a READ or WRITE routine, a CHECK routine, and a routine to service the NOTE/POINT macro instruction if it is specified.

- It loads appendages and puts their addresses in the DEBAVT.
- It loads the end-of-block routines specified by the stage 2 executors.
- It issues a DMABCOND macro instruction if LRECL=X is specified and the processing mode is not Locate.
- It issues a DMABCOND (abend code 013-4) macro to cause an abend if ISO/ANSI/FIPS records are to be processed and there is no record area present.

- It places the IDs of the routine, loaded into the DEB subroutine ID field, and the addresses of the routines into the DCB.
- It specifies in the WTG table that executor IGG0191I is the next executor required for this DCB.
- It searches the WTG table to determine to which executor it should pass control.

Stage 2 and 3 OPEN Executor IGG0193B: Executor IGG0193B is entered after executor IGG0193I if the DCB specifies:

Direct access

This executor operates as follows:

- This is the stage 2 and 3 open executor for non-WRITE load direct-access processing. Each DCB that has IGG0193B in its where-to-go (WTG) table entry is processed by the module's base routine, OPENEXEC.
- It gets a storage area for, and initializes, the SAMB control block (SAMBBLD). The amount of space of CCWs, count fields, and IDAWs (indirect address words) is calculated from DCBBLKSI (if not zero), and DCBBUFNO for SQAM or DCBNCP for BSAM.
- It initializes the key zero IOB in the SAMB (IOBBLD).
- It gets a storage area for, and initializes, the ICQE and caller key IOBs. It initializes the caller key IOBs by copying the key zero IOB into them (ICQIOBLD).
- It initializes the DEBAVT (AVTBLD).
- It initializes the fields in the DCB. It calls subroutines to load the front end routines and places their addresses in the DCB (DCBBLD).
- It primes the buffers for QSAM input and update (BFRPRIME).

Stage 3 OPEN Executor IGG0198L (SYSIN/SYSOUT): IGG0198L receives control after the SAM-SI OPEN executor IGG0199W.

The executor operates as follows:

- It determines which processing modules are required to process the SYSIN or SYSOUT data set.
- If BSAM is specified in the DCBMACRF field of the DCB, the BSAM processing module, IGG019DK, is loaded into virtual storage. If input is also specified, module IGG019BB is also loaded to process the CHECK macro instruction. Otherwise, IGG019DK also handles the CHECK macro instruction.
- If QSAM is specified, the QSAM CI processing module IGG019DJ is loaded into virtual storage.
- If input is specified, module IGG019AQ is also loaded to process an end-of-data condition.
- It sets the CI bit in the DCBCIND1 field to indicate that this DCB is processed by the SAM-SI routines.
- It marks the current entry in the WTG table to indicate that no further executor processing is required for this DCB.
- It refreshes the processing program's DCB from the copy maintained by the open routines.
- It then searches the WTG table to determine whether to give control to another executor, or branch back to itself. If

there are no other entries in the WTG table, the executor returns control to the open routines.

CLOSE EXECUTORS

Figure 25 on page 141 shows the conditions that cause the CLOSE executors to gain control. IGG0201A or IGG0201Z receives control if one of the sequential access methods is used. Control goes to IGG0201A if the device type is tape or unit record. Executor IGG0201X is an extension of IGG0201A. If the device type is direct-access storage, control is passed to IGG0201Z. Executor IGG0201B receives control after executors IGG0201A or IGG0201Z if QSAM was used with an output data set and a channel program encountered an error condition while one of the other CLOSE executors had processor control. Executor IGG0201P receives control from IGG0201A whenever the 3525 or the 3505 with OMR or RCE is specified. Executor IGG0201R is an extension of IGG0201P. Executor IGG0201W receives control whenever a SYSIN or SYSOUT data set is being processed.

Control returns to the CLOSE routine of I/O support when CLOSE executor processing is completed.

Access Method Options	Selections					
Tape or unit record	X	X	X		X	
Direct-access storage			X	X		
Permanent error or end-of-volume condition when using QSAM for output (tape, DA only)		X		X		
*, DATA, or SYSOUT specified on DD statement						X
3505 (OMR/RCE) or 3525					X	
Executors						
IGG0201A	1A	1A			1A	
IGG0201B		1B		1B		
IGG0201P					1P	
IGG0201R					1R	
IGG0201W						1W
IGG0201X	1X	1X				
IGG0201Y			1Y	1Y		
IGG0201Z			1Z	1Z		

Figure 25. CLOSE Executor Selector

CLOSE Executor IGG0201A: IGG0201A receives control from the CLOSE routine of I/O support if the DCBDSORG field specifies a value of PS and if the device type is tape or unit record.

The executor operates as follows:

- It turns on the CLOSE-in-process bit in the DCB.
- If the 3525 or the 3505 with either OMR or RCE is specified,

the executor specifies in the WTG table that executor IGG0201P is required for this DCB.

- For QSAM output on 2540 devices, it issues EXCP macro instructions to punch two blank cards to allow the ERPs to gain control when an error occurred on either of the two last cards punched.
- For QSAM input or BSAM data sets, a PURGE macro instruction is issued.
- If the OPEN parameter is output and the DCB specifies QSAM, the executor issues a TRUNC and, if the processing mode is Locate, a PUT macro instruction to cause scheduling of the last buffer. On return of control, the executor awaits execution of the last channel program. The TRUNC and PUT routines are entered via the SYNCH SVC if the user's key is greater than 7.
- If all channel programs were executed without encountering either an end-of-volume condition or a permanent error, the executor continues processing.
- For magnetic tape devices, if any of the preceding channel programs encountered 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.
- For printers, if the clear printer is a valid command, clear printer/nop commands are issued to ensure all data has been printed. For other printers, issue a WRITE NOSPACE command to clear the printer print line buffer.
- It sets an audit trail bit to indicate that a PURGE has been done. These bits have meaning only during force close processing. The audit trail is passed to a user's STAE routine.
- It sets up the WTG table to pass control to IGG0201X.
- It then searches the WTG table to process another DCB or pass control to another executor.

CLOSE Executor IGG0201B (Error Processing): IGG0201B receives control after either executor IGG0201A or IGG0201Z if one of the latter finds that a channel program for an output data set using QSAM encountered a permanent error or an end-of-volume condition.

The executor operates as follows:

- It determines whether a channel program encountered a permanent error or an end-of-volume condition.
- If a permanent error occurs for a direct-access device, it enters the track balance routine to get the bad record erased.
- If a channel program encountered an end-of-volume condition, the executor finds the IOB associated with that channel program and issues an EOV. 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 it did when it first received control.
- If the DCB specifies either a DCBDSORG field value of PO or POU with a DD statement of the form DSNAME (MEMBERNAME) the executor issues a STOW macro instruction. On completion of the STOW routine, the executor tests for errors, such as

insufficient space in the directory. For any type of error, the executor issues an DMABCOND macro instruction.

- The executor specifies in the WTG table that the next executor needed for this DCB is either IGG0201Y for direct-access devices or IGG0201X for all other devices.
- It then searches the WTG table to either process another DCB or to pass control to the next module.

CLOSE Executor IGG0201P: This module receives control from IGG0201A whenever:

The 3525 is specified or the 3505 is specified with either OMR OR RCE.

The module operates as follows:

- It turns on the CLOSE-in-process bit in the DCB.
- Tests are made to determine if either OMR or RCE is being used with the 3505.
- If either is being used, the module issues a feed and stacker-select command (with the OMR/RCE flag bit off) to return the device to normal punched mode.
- If either an associated data set or PRINT is being used with the 3525, the following apply:

File Type	Feed Caused by Close of
Print	Print File
Read/print	Read File[1]
Read/punch/print	Read File[2]
Read/punch	Read File[2]
Punch/print	Punch File
Punch/interpret	Punch File
Read	Read File
Punch	Punch File

Notes:

1. A feed is executed if an end-of-file is caused by the hardware; a feed is not executed if it is caused by a data delimiter card.
2. Punching or printing delimiter cards is not allowed for these file types, because the CLOSE routine always issues a feed command.

- If a channel program for an output (QSAM) data set encountered a permanent error, IGG0201B is specified in the WTG table as the next executor required for this DCB. Otherwise, executor IGG0201R is specified in the WTG table.

  It then searches the WTG table to pass control to another executor.

CLOSE Executor IGG0201R: This module receives control from IGG0201P.

The module operates as follows:

- It frees buffer space from the buffer pool.
- It also frees IOB and ICB space.
- It clears BSAM and QSAM vectors in the DCB.
- It specifies in the WTG table that executor IGG0201B is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

CLOSE Executor IGG0201W (SYSIN/SYSOUT): Executor IGG0201W receives control if the CLOSE routine (see Diagram L) determines that the SAM-SI CLOSE executor is required to process a DCB for a SYSIN or SYSOUT data set.

The executor operates as follows:

- It constructs a CLOSE parameter list for the ACB built by the SAM-SI OPEN executor for this DCB.
- If QSAM PUT locate mode is specified, a final PUT macro instruction is issued to clear the I/O area.
- It deletes the BSAM (IGG019DK and IGG019BB) or QSAM (IGG019DJ and IGG019AQ) processing modules loaded by the OPEN executor, IGG0198L. The processing modules that handle CI and SAM requests (IGG019BB and IGG019AQ) are not deleted if concatenation is in process.
- It issues a CLOSE macro instruction for the ACB.
- It issues a FREEMAIN macro instruction for the area occupied by the JES compatibility interface control block (CICB) and the record area obtained for collecting BSAM variable spanned segments.
- It searches the WTG table to pass control to another executor.

CLOSE Executor IGG0201X: Executor IGG0201X is a continuation of executor IGG0201A and receives control from that executor or from IGG0201B if an EOV condition arose during processing in IGG0201A.

The executor operates as follows:

- For QSAM:

  It frees the record area if it was gotten by the OPEN executors.

  It frees the buffers gotten by the OPEN executors if concatenation of unlike attributes was specified.

  It returns the buffer to the buffer pool for all other conditions.

- 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 by using a FREEMAIN macro instruction.
- It sets audit trail bits to indicate what processing was done. These bits have meaning only during force close. The audit trail is passed to a user's STAE routine.
- 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 processes

another DCB, returns control to the CLOSE routines, or, if force CLOSE is in control, returns to the SAM force CLOSE executor, IGG020T1, with a return code of 0 in register 15.

CLOSE Executor IGG0201Y: IGG0201Y receives control from executor IGG0201Z or from IGG0201B if an EOV or permanent error was detected by IGG0201Z.

The executor operates as follows:

- When record-ready channel programs are constructed, a GETMAIN macro instruction is issued for more bytes during open IOB construction. In the CLOSE routine, when the IOB and channel program areas are freed, the number of additional bytes is computed and added to the byte count before issuing the FREEMAIN macro instruction.
- It frees the segment work area for a DCB that specifies BFTEK=R, RECFM=VS, and MACRF=WL.
- It returns buffers to the buffer pool if they were gotten by open executors.
- It frees the SAMB, ICQE, and IOBs if they were gotten by the OPEN executor. It also zeros the addresses of these control blocks in the DCB.
- It frees the buffers if concatenation of unlike attributes was specified.
- It frees the record area obtained by an OPEN operation when a DCB specifies BFTEK=A, spanned record, and QSAM locate mode.
- The executor specifies in the WTG table that processing for this DCB is completed. Depending on the remaining entries in the WTG table, it then processes another DCB, returns control to the common close routines or, if force CLOSE is in control, returns to the SAM force CLOSE executor, IGG020T1, with a return code of 0 in register 15.

CLOSE Executor IGG0201Z: Executor IGG0201Z receives control from the close routine of O/C/EOV if the DCBDSORG field specifies a value of PS or PO and if device type is direct-access storage.

The executor operates as follows:

- If the task is abnormally terminating, the following processing takes place.
  - No executor processing is performed if the data set is open for input or update, or the last operation was not a write, or the DEB extension indicates an OPEN/CLOSE/EOV ABEND occurred.
  - For partitioned data sets, DCBFDAD is set from DCBRELAD. SVC 25 is issued and no further executor processing is performed.
  - For sequential data sets processed by BSAM, DCBFDAD is set from the IOBSEEK field in the SAMBIOB. SVC 25 is issued and no further executor processing takes place.
  - For sequential data sets processed by QSAM, normal executor processing listed below is performed. Purged I/O is restarted.
- If the OPEN parameter is OUTPUT and the DCB specifies QSAM, the executor issues a TRUNC and, if in locate processing mode, a PUT macro instruction to cause scheduling of the last buffer. On return of control, the executor awaits execution of the last channel program.

- For QSAM input or BSAM data sets, a PURGE macro instruction is issued.
- 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 OUTPUT and either a DCBDSORG field value of PO, or WRITE or PUT with a DD statement of the form DSNAME (MEMBERNAME) is 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 a DMABCOND macro instruction.
- The executor specifies in the WTG table that module IGG0201Y is the next executor for this DCB. Depending on the remaining entries in the WTG table, it then either processes another DCB or transfers control to the next module.

FORCE CLOSE EXECUTORS

SAM-SI Force CLOSE Executor IGG020FC: Executor IGG020FC receives control from the O/C/EOV force CLOSE executor module, IFG0RR0B, when it determines that DCBs under JES control must be closed. The executor frees resources acquired for opened or partially opened SYSIN and SYSOUT DCBs that are being forced to a closed status. It provides as much of the normal close functions as possible in restoring the DCB to its preopen condition.

The executor locates the CICB and performs the following operations:

- Issues a CLOSE macro instruction for the ACB contained in the CICB.
- Frees the record area for variable-length spanned records.
- Deletes any processing modules loaded for this DCB.
- Frees the storage obtained for the CICB.
- Returns control to the calling routine.

If the failure occurs during open processing and the CICB was not created, no further processing is required and control is returned to the calling routine, with a return code of 0.

If the CICB cannot be located because the error occurred during other than open processing, control is returned to the calling routine, with a return code of 4.

Force CLOSE Executor IGG020T1: Executor IGG020T1 receives control from IFG0RR0B during force close processing for SAM, PAM, or DAM data sets. The primary function of the force CLOSE executor is to free resources associated with the DCB.

The executor operates as follows:

- If the error occurred during close for QSAM or BSAM output DCBs, DCBFDAD is set from the IOBSEEK field in the SAMB IOB for sequential data sets, and from DCBRELAD for partitioned data sets. SVC 25 is issued using the SAMB JOB.

- If the error occurs during open processing and the user's copy of the DCB has not been updated by the OPEN executors, the following actions are taken:

For SAM or PAM:

- It frees a logical record area if obtained by OPEN executors.
- It frees the buffer pool if the user's buffers were gotten by the OPEN executors and concatenation of unlike attributes was specified; otherwise, it returns the buffers to the buffer pool.
- It frees the IOBs and ICBs and their channel programs if they were gotten by the OPEN executors.
- It frees the segment work area if it was gotten by OPEN executors.
- It frees the SAMB if it was obtained by the OPEN executor.
- It deletes the message CSECT if it was loaded by the OPEN executors.
- It deletes any UCS and FCB images loaded.
- It issues a CLOSE IMGLIB SVC for SYS1.IMAGELIB.

For BDAM:

- It frees the buffers.
- It frees the unscheduled list if it exists.
- It frees the segment work area if it was gotten by the OPEN executors.
- It frees the READX list if it was gotten by the OPEN executors.

The force CLOSE executor then returns to common CLOSE with a return code of zero in register 15.

If the error occurs during open processing and the user's DCB was refreshed from the protected DCB, the force CLOSE Executor sets up a retry address at RRXRETRY and attempts to execute the normal CLOSE executor string. It also issues a FREEMAIN macro instruction for the register save area gotten by IGG01911 when priming QSAM input buffers.

- If normal CLOSE processing is successful, the CLOSE executor, upon detecting a force CLOSE entry, returns to this force CLOSE executor with a return code of zero in register 15.
- If normal CLOSE processing is not successful, the second level recovery routine of O/C/EOV gives control to the address specified in RRXRETRY. The force CLOSE executor then moves the audit trails to the component recovery status a ea (CRSA) with a return code of 8 in register 15.

If the error occurs during other than open processing, the force CLOSE executor returns to the common close recovery routine with a return code of 8 in register 15.

BUFFER-POOL MANAGEMENT

Buffer-pool management routines form virtual storage space into buffers, and return buffers that are no longer needed. Figure 26 lists the buffer-pool management routines.

Type	Module Name	Function
GETPOOL	IECQBFG1	This routine obtains virtual storage and forms a buffer pool.
BUILD	IECBBFB1	This routine forms a buffer pool in virtual storage supplied by the processing program.
GETBUF	(Macro Expansion)	This routine provides buffers from the buffer chain.
FREEBUF	(Macro Expansion)	This routine returns buffers to the buffer pool.
FREEPOOL	(Macro Expansion)	This routine returns virtual storage previously used for a buffer pool.
BUILDRCD	IGG019B0	This routine allows a pointer to a record area to be incorporated in a buffer pool in virtual storage supplied by the processing program.

Figure 26. Buffer-Pool Management Routines

GETPOOL Module IECQBFG1: Module IECQBFG1 obtains virtual-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 doubleword multiple if the specified length is not such a multiple.
- It determines buffer alignment from the DCBBFALN 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 8 bytes of storage obtained.
- If doubleword (not fullword) alignment is specified in the DCBBFALN field in the DCB, the module starts the first buffer at the byte immediately following the BUFCB.
- If fullword (not doubleword) alignment is specified in the DCBBFALN 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 to 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 27 on page 149 illustrates the buffer-pool control block (BUFCB)

that describes the buffer pool. Figure 28 on page 149 illustrates the buffer-pool structures formed by the GETPOOL module.

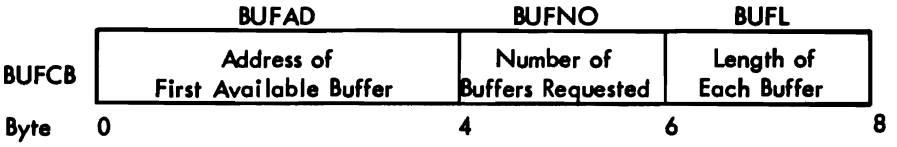


Figure 27. Buffer-Pool Control Block

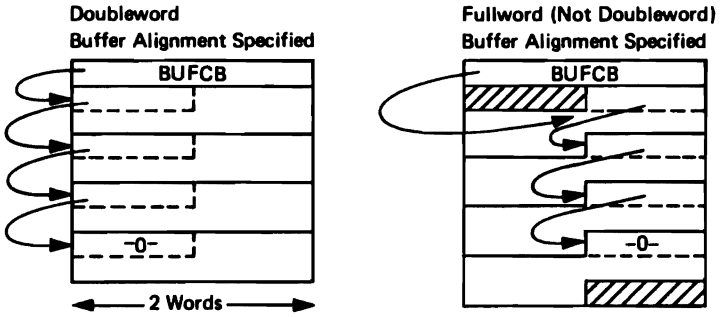


Figure 28. GETPOOL Buffer-Pool Structures

BUILD Module IECBBFB1: Module IECBBFB1 forms virtual 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 fullword multiple if the specified length is not such a multiple.
- It constructs a buffer-pool control block in the first 8 bytes of the virtual-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.

Figure 29 on page 150 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 27 illustrates the buffer pool control block (BUFCB), Figure 30 on page 150 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 inline 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 inline 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.

Alignment of first byte of space passed in BUILD macro instruction	Parity of number of words in buffer length after rounding up length parameter macro of BUILD macro instruction	Buffer pool structure
Doubleword	Even	A
	Odd	B
Fullword	Even	C
(Not doubleword)	Odd	D

Figure 29. Build Buffer-Structuring Table

FREEPOOL Macro Expansion: The purpose of this coding is to return the space previously allotted to the buffer chain to available virtual storage. The macro expansion produces inline 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.

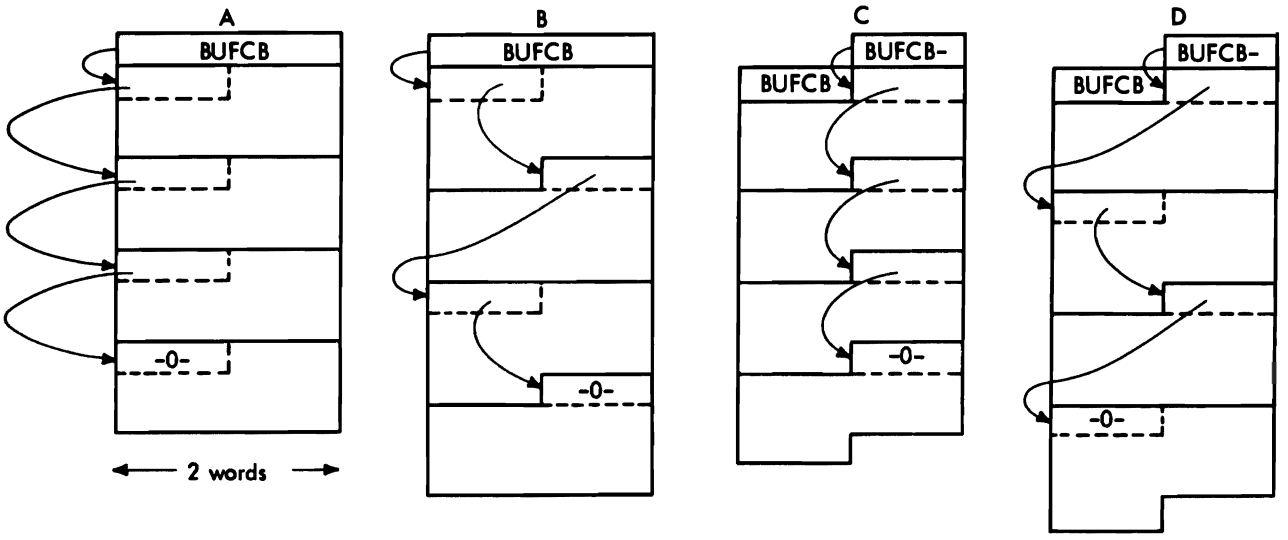


Figure 30. Build Buffer Pool Structure

BUILDRCD Routine IGG019B0: This routine forms virtual-storage space supplied by the processing program into buffers and links the buffer pool to a record area also supplied by the processing program. 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 fullword multiple if the specified length is not such a multiple.
- It constructs a buffer-pool control block (see Figure 33 on page 170) in the first 12 bytes of the virtual-storage space provided by the processing program.
- It turns on the high-order bit of the BUFLG byte of the buffer-pool control block to indicate that a record area address is present.
- It clears the control field (32 bytes) of the record area.
- It stores the record area length in the record area (see Figure 34 on page 241) provided by the processing program.
- 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 next buffer is chained to the preceding buffer until all buffers are built.
- It returns control to the processing program.

Figure 31 on page 151 illustrates the buffer-pool control block (BUFCB) that describes the buffer pool when logical record interface is required for variable-length spanned records processed in the locate mode.

Figure 32 illustrates the record area used to assemble and segment a spanned record. This record area is either acquired dynamically by data management at OPEN time, when the DCB specifies RECFM-VS/VBS, MACRF=GL/PL, and BFTEK=A, or provided by the problem program by means of a BUILDRCD macro instruction.

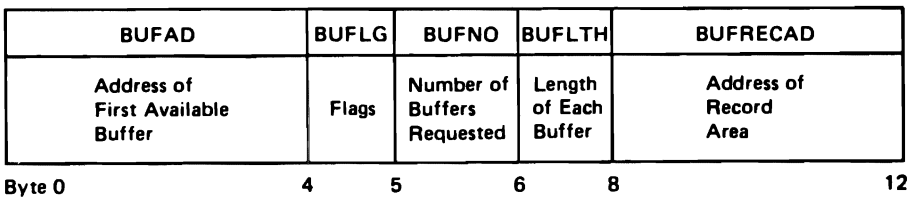


Figure 31. Buffer-Pool Control Block

BUFAD 4 bytes; contains the address of the first available buffer in the pool.

BUFLG 1 byte; set to X'C0' when a record area address is present in the buffer control block.

Bit	Meaning
0-1	Record area present
1-1	Buffer control block extended
2-7	Reserved

BUFNO 1 byte; contains the number of buffers requested.

BUFLTH 2 bytes; contains the length, rounded to the nearest fullword of each buffer requested.

BUFRECAD 4 bytes; contains the starting address of the record area.

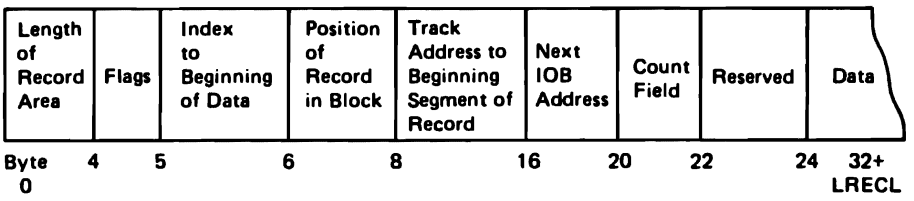


Figure 32. Record Area Used to Assemble and Segment a Spanned Record

A description of the fields contained in the record area follows:

- **Length of Record Area.** This 4-byte field contains the length of the entire record area (data field + 24 bytes). The length may be determined by the LRECL of the DCB macro at OPEN time plus 8 bytes for alignment or it may be specified in the length of the record area parameter of the BUILDRCD macro instruction, in which case the BUILDRCD routine places the length of the record area in this field. If extended logical record interface (XLRI) is used, the DCB LRECL value is a multiple of 1024, which is used to calculate the requested record area size. The second bit of the first byte of this field is set on by the COBOL processor to indicate special processing of variable-length spanned records. If this bit is set, all records (spanned or nonspanned) are presented to the processing program in the record area.
- **Flags.** This 1-byte field is used for internal data management control flags.
- **Index to Beginning of Data.** This 1-byte field contains the index value to the beginning of the data (record descriptor word) in the data field.
- **Position of Record in Block.** This 2-byte field contains the relative position of the beginning segment of a record within the block.
- **Track Address to Beginning Segment of Record.** This 8-byte field is used to save the track address of that block that has a beginning segment of a record that is being processed. The low-order 3 bytes of this field are used to save the record address of the block that will have the beginning segment of a record if a spanned record is to be written.
- **Next IOB Address.** This 4-byte field is used to save the next IOB address if a spanned record is to be written.
- **Count Field.** This 2-byte field is used to accumulate the number of bytes of data moved while segmenting. For extended logical record interface (XLRI), the count field is four bytes long and includes the two bytes of the following field.
- **Reserved.** Not used. If extended logical record interface (XLRI) is used, this field is used as part of the count field.

- **Data.** The assembled logical record is located in this field. The maximum length of this field is either determined by the LRECL field of the DCB macro at OPEN time plus 8 bytes for alignment or equal to 24 bytes less than the length of the record area parameter of the BUILDRCD macro instruction.

PROBLEM DETERMINATION

Problem determination assists the user in determining the causes of abends by providing more information as to the cause of the abnormal termination. The recording and making available of significant information about the problem may eliminate the need for a storage image dump. Better abend interpretation will be possible with the following problem determination operations:

- Write-to-programmer giving the abend code, a return code that further describes the abend condition, and job environment information.
- Recording of all control blocks relevant to the abend condition on a GTF data set, which will be dumped automatically by ABDUMP, or at the user's initiation by AMDPRDMP.
- A user abend exit is provided to allow the evaluation of the condition before the abend is taken.
- An abend that provides a dump of relevant control blocks.

Problem determination is of particular benefit in the OPEN executors because having an alternative to an immediate abend results in greater latitude in the control of the termination of a task. The error can be evaluated and the need for that data set at the time the error occurred can be determined, with the option to continue processing without it.

Problem Determination Module IFG0559C: Module IFG0559C traces the data associated with a particular abend.

The module operates as follows:

- It receives control through an XCTL macro instruction from the O/C/EOV problem determination module, IFG0559B, when it senses a SAM problem determination flag.
- It issues a MODESET macro instruction to change to the key of the caller.
- It issues a GETMAIN macro instruction for work area storage.
- If the GETMAIN macro instruction is successful, it issues a GTRACE macro instruction to record, in the GTF data set, the data associated with the abend designated by the abend and condition codes. In addition to the TIOT DDNAME and the abend condition code, which are always present, one or more of the following data areas is traced:

  DCB for BSAM or QSAM

  DECB for BSAM only

  Track capacity - maximum block size

  Current DEB extent entry

  All DEB extent entries

  IOB or ICB seek field

  First 88 bytes of the BDW and the block currently being processed

First 88 bytes of the RDW and the record currently bring processed

The following is a list of abends, their associated condition codes, and the data traced for each.

ABEND Code	Condition Code	Areas Traced
002	04	DCB, IOB or ICB seek field, record
002	08	DCB, DECB, block
002	0C	DCB, DECB, maximum block size, block
002	10	DCB, DECB, block
002	14	DCB, DECB, block
002	18	DCB and record
002	1C	DCB, DECB, maximum block size, block
002	20	DCB, DECB, current DEB extent, maximum block size, block
002	24	DCB, DECB, current DEB extent, maximum block size, block
008	04	All DEB extents, block

- If the GTRACE macro instruction is successful, a LOAD macro instruction is issued to load the message CSECT, IGGMSG01. A WTO macro instruction is issued to inform the programmer that the GTF data set contains records associated with this abend. Upon return, a DELETE macro instruction is issued to delete the message CSECT.
- It issues a FREEMAIN macro instruction to release the work area storage.
- It transfers control to module IFG0559E upon successful completion or if an error occurred in the GETMAIN or GTRACE macro instruction.

SVC ROUTINES

SVC routines are used when the process requires operation in the supervisor state. The functions provided are ones that cannot be done in the problem state or in the user's key.

DEVTYPE ROUTINE

DEVTYPE SVC Routine IGC0002D: This routine locates and passes to the requestor the characteristics of the device specified in the DD statement. The module operates as follows:

- It issues an ESTAE macro instruction to establish a task recovery routine, IGCT002D, to intercept abnormal terminations.
- It searches the UCB, its DASD class extension table, and the device characteristics table for the required information.
- For the 3340 and the 3380 (all models), it determines the number of cylinders on the pack.

- It places the data in the output area and returns to the calling program.

IMGLIB ROUTINE

IMGLIB SVC Routine IGC0010E: The IMGLIB SVC routine, IGC0010E, builds a skeleton DCB and DEB for the SYS1.IMAGELIB data set or deletes the DCB and DEB for the SYS1.IMAGELIB data set, depending on the parameter passed to it in register 1. The routine is entered from the SVC 105 instruction.

The IMGLIB macro is issued by OPEN executors and by SETPRT routines and can be issued by users.

The routine operates as follows:

- It issues an ESTAE macro instruction to establish a TRR, IGCT010E, to intercept abnormal terminations.
- It makes a test to determine whether the control blocks for IMAGELIB need to be built or deleted. If register 1 contains 0's, a DCB and DEB are built.
- It uses a GETMAIN macro instruction to obtain a work area and then uses a LOCATE macro instruction to determine where the IMAGELIB volume is residing.
- It takes the address of the UCB table from the CVT and searches for the corresponding UCB.
- It uses the OBTAIN macro instruction to read in the format-1 DSCB and uses the information read and the UCB address to construct a skeleton DCB and DEB for the SYS1.IMAGELIB volume. The format-1 DSCB describes up to three extents. The SYS1.IMAGELIB data set can reside on up to 16 extents on a permanently resident volume.
- If there are more than three extents on SYS1.IMAGELIB, the format-3 DSCB seek address is obtained from the format-1 DSCB. It uses the OBTAIN macro to read in the format-3 DSCB and uses the information read and the UCB address to construct additional DEB extent descriptions.
- If register 1 contains an address when the routine tests to determine whether the control blocks for the IMAGELIB volume need to be built or deleted, the DCB and DEB for IMAGELIB are to be deleted.
- It uses the FREEMAIN macro instruction to delete the control blocks. If the DEB is not on the DEB chain or it does not point back to the DCB, a 169 abend is issued.
- It returns control to the calling routine through a BR 14 instruction.

TRACK BALANCE, TRACK OVERFLOW ERASE, DEB/SAMB UPDATE ROUTINES

Control Module IGC0002E (SVC 25—Track Balance, Track Overflow Erase, DEB/SAMB Update): Module IGC0002E consists of a track balance routine, a track overflow erase routine, and an update routine. The track balance routine calculates the available space at the end of a track and erases the end of the track. The track overflow erase routine erases the end of a track and perhaps several full tracks. The update routine either updates the SAMB's IOBSEEK field or initializes the DEB's DEBBLKSI field. The track balance routine determines the available space by reading the count fields of the records on the track and 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 entered when SVC 25 is executed by the following modules:

READ/WRITE module IGG019BA (track balance)

End of Block module IGG019TV (update DEBBLKSI)

NOTE/POINT module IGG019BK (update SAMB)

CLOSE executor IGG0201B (track balance)

- It issues an ESTAE macro instruction to establish a TRR, IGCT002E, to intercept abnormal terminations.

Track Balance Routine: This routine is given control when register 1 on input is not negative and the track-overflow flag DCBCNTOV in DCBCIND1 is zero. 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 macro instruction.

The routine operates as follows:

- DCBFDAD is established as the record preceding the next record to be written.
- If the record number in DCBFDAD is zero, the whole track is available. DCBTRKBAL is set equal to the track capacity of the device (from the device characteristics table) and control is returned to the caller.
- A work area is obtained, a channel program is built, and an EXCP is issued to read the count fields (CCHHRKDD) of each record on the track, up to the number indicated by DCBFDAD. The channel program ends with an erase CCW, which erases all records following the last record (as indicated by DCBFDAD).
- It determines the exact track balance by subtracting the sum of all key-lengths and data lengths in all the count fields and the not-last record overheads from the track capacity and inserts the difference in the DCBTRBAL field of the DCB.

Track-Overflow Erase Routine: This routine is given control if register 1 on input is not negative and the track overflow flag (DCBCNTOV in DCBCIND1) is one. 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.

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 write routine, irrespective of any errors in the execution of the channel program.

DEB/SAMB Update Routine: This routine is given control if register 1 is negative. The DEB is validated by branching to the DEBCHK routine. If register 0 is zero when SVC 25 is entered:

- The DEBBLKSI field is initialized as the sum of key length and block size in the DCB.

If register 0 is four when SVC 25 is entered:

- The IOBSEEK field of the IOB in the SAMB is set equal to DCBFDAD.

BSP ROUTINE

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 verifies that the passed DCB describes a magnetic tape or direct-access device data set, and that the data set is being processed by BSAM. To accomplish this, the module operates as follows:

- It receives control after it is loaded.
- It issues an ESTAE macro instruction to establish a TRR, IGCT0069, to intercept abnormal terminations.
- It issues a MODESET macro instruction to change to the key of the caller.
- If the DCB is being processed by the CI and if a CI backspace routine entry point is provided, it gives control to the CI routine. When the CI routine relinquishes control, or if no CI routine is provided, it returns control to the processing program.
- If the device is a terminal, it returns control to the processing program.
- If a dummy data set is being processed, it returns control to the processing program.
- If the device type is not magnetic tape or direct access, reason and return codes are put in registers 0 and 15 and control is returned to the caller.
- If either a tape mark or a direct-access EOF was read, reason and return codes are put in registers 0 and 15 and control is returned to the caller.
- It issues a GETMAIN macro instruction to obtain storage in which to build an IOB, an ECB, and a channel program.
- It builds and initializes an IOB and an ECB.

From this point on, the control path depends upon the type of I/O device.

For magnetic tape, the module operates as follows:

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

- It issues a FREEMAIN macro instruction to free the work area.

- It issues a MODESET macro instruction to return to KEY 0 and returns control to the caller.

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

- It decreases 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 DCBFDAD value has been established, the module updates the DCBFDAD value. It also updates the IOBSEEK field in the SAMB's IOB and sets the SAMSCHPR flag in the SAMB. Next, the module updates the IOBSEEK field in the next user IOB to be scheduled, or in all user IOBs if an extent boundary was crossed.

- If a track (or cylinder or extent) boundary is backspaced over, all records' counts on the track backspaced to are read. The last record's count field is used to identify the last record on the track.

- If there is no valid preceding DCBFDAD value because the processing program has attempted 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.

- It issues a FREEMAIN macro instruction to free the work area.

- It issues a MODESET macro instruction to return to KEY 0 and returns control to the caller.

STOW ROUTINES

STOW Module IGC0002A (SVC 21): Module IGC0002A builds the control blocks, buffers, and channel program required to perform the requested function and to do the diagnostics to verify the validity of the caller's request.

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.

The module operates as follows:

- It receives control when it is loaded.
- It issues an ESTAE macro instruction to establish a TRR, IGCT0021, to intercept abnormal terminations.
- It issues a MODESET macro instruction to change to the key of the caller.
- If the DCB is neither OPEN nor OPEN for input, a MODESET macro instruction is issued to return to key 0, reason and return codes are put into registers 0 and 15, and control is returned to the caller.
- It issues a GETMAIN macro instruction to obtain storage for a work area in which to save information about the function being performed; save the parameters supplied by the caller; and build an IOB, ECB, channel program, and three buffers used in reading and writing directory blocks.
- If the GETMAIN macro instruction is not successful, a MODESET macro instruction is issued to return to key 0, reason and return codes are put into registers 0 and 15, and control is returned to the caller.
- It initializes the channel program and issues an EXCP macro instruction to search the directory for an entry block with a key equal to or higher than the member name. It reads that and the next entry block into the input buffers. For the change option (C), the search is on the member name that is the lowest in alphameric sequence.
- It checks the validity of the option requested, as follows:

  Add. Verifies that the member name does not already exist.

  Replace. Verifies that the member name exists and, if not, sets the return code and changes the option to Add.

  Change. Verifies that the member name to be changed exists and that the new member name does not duplicate an existing name.

  Delete. Verifies that the member name exists.

- If an I/O error occurs while directory entry blocks are being read or if the option requested is invalid, a FREEMAIN macro instruction is issued to free the work area, a MODESET macro instruction is issued to return to key 0, reason and return codes are put into registers 0 and 15, and control is returned to the caller.
- If the option requested is valid, the module transfers control to module IGG0210A through an XCTL macro instruction.

STOW Module IGG0210A: Module IGG0210A builds the channel program used by module IGG021AB and, if required, writes an EOD marker following the last member written.

The module is loaded and receives control through an XCTL macro instruction issued by module IGC0002A.

The module operates as follows:

- It writes an EOD marker following the last member written if the following conditions are met:

  The Add or Replace option was specified.

  The entry being added or replaced is not an alias.

The DCB was not opened for RDBACK or UPDATE.

The last I/O operation was write.

- If the data set must be extended to write the EOD marker, the module issues a MODESET macro instruction to change to key 0, a SETLOCK macro instruction to obtain the local lock, and branches to DEBCHK to check the validity of the caller's DEB. If the DEB is valid, the DEBVOLSQ is changed for EXCP, a SETLOCK macro instruction is issued to free the local lock, and a MODESET macro instruction is issued to return to the caller's key.

  It then changes the MACRF operand in the user's DCB to EXCP and issues an EOV that points to that DCB.

  Upon return, it restores the MACRF operand and validates and restores the DEBVOLSQ field.

  If the DEB is not valid, it issues an ABEND macro instruction to terminate processing.

- If an EOD marker is written after the last member, the FDAD, RELAD, and TRBAL fields in the caller's DCB are updated.

- If an I/O error occurs while the EOD marker is being written, the module frees the work area, returns to key 0, sets the reason and return codes in registers 0 and 15, and returns control to the caller.

- It returns the TTR of the last member written if the following conditions are met:

  The add or replace option was specified.

  The entry being added or replaced is not an alias.

- It builds the channel program used by module IGG021AB to read and write directory blocks.

- If no errors are detected, it transfers control to module IGG021AB through a XCTL macro instruction.

STOW Module IGG021AB: Module IGG021AB maintains partitioned data set directories in ascending order of the binary values of the names of the members.

Module IGG021AB is loaded and receives control through an XCTL macro instruction issued by module IGG0210A.

The module operates as follows:

- If the option requested is add, replace, or change and if there are no unused directory blocks, a dry run on the directory is made to determine if sufficient space is available in which to perform the requested function.

- It adds, replaces, changes the name of, and deletes directory entries, per the requested options, by issuing an EXCP macro instruction to write and read directory blocks.

- It expands or compresses the directory as necessary to accomplish the requested function.

- If an I/O error occurs while writing or reading directory blocks, or, if there is not sufficient space remaining in the directory, processing in this module is terminated.

- It issues a FREEMAIN macro instruction to free the work area.

- It issues a MODESET macro instruction to return to key 0.

- It returns control to the calling program.

FIND (C Option) Macro Expansion: The macro expansion moves the relative address (TTRK) from the passed input parameter list to the DCBRELAD field in the requester's DCB. The FIND macro instruction then does a branch-and-link to the POINT routine.

FIND (D option) Macro Expansion: The macro expansion loads the DCB address into register 1 and complements it to notify BLDL this is a FIND request. The address of the BLDL parameter list (that is, the list of member names) is loaded into register 0. SVC 18 (BLDL) is then issued.

BLDL Macro Expansion: This is the same as for the FIND (D option) described above, except that the DCB address is not complemented.

Resident Module IGC018 (BLDL): The module gains control through an SVC 18 instruction in a processing program. A FIND (D option) or BLDL macro instruction expansion generates an SVC 18 instruction, which causes control to pass to CSECT IGC018. BLDL is link-edited into the nucleus as an RSECT (read only). It executes in 31-bit addressing mode, RMODE ANY, and supports 31-bit input parameter lists.

A second csect in module IGC018, named IEC0SCR1, contains the system convert routines and is link-edited into the nucleus as RMODE 24, AMODE ANY. This means it resides below the 16-megabyte virtual storage line and executes in the addressing mode (24 or 31 bit) of the caller.

Programs may use a BALR instruction and the address found in the communication vector table (CVT) for entry points IECPCNVT(CVTPCNVT), IECPRLTV(CVTPRLTV), and IEC0SCR1(CVT0SCR1), to pass control to the respective convert routines.

The BLDL routine operates as follows:

- It issues an FESTAE macro instruction to establish the recovery routine.
- Based on the key in the DEB (pointed to by the passed or defaulted DCB) a GETMAIN is issued for a work area in subpool 253 for a key 0 data set or subpool 230 for a nonzero key data set. Among the contents of the work area are the parameter list used if the LNKLST lookaside table is available, and the IOB, channel program, and buffers if I/O is required. The work area must be in the same key as the DCB and DEB.
- If the LNKLST lookaside table is available, it is searched by BLDL for any of the following conditions:
  - The passed DCB is the LINKLIB DCB (same address as in CVTLINK).
  - The DCB address is zeros and there is no steplib, tasklib, or joblib specified.
  - The specified member(s) are not found in the specified data sets.

  Note: The LNKLST lookaside (LLA) directory replaces the resident BLDL, table which is initialized during NIP. The LNKLST lookaside directory is a resident directory of entries built during IPL by starting the LLA procedure. It contains the directory entries for all members in SYS1.LINKLIB and for all data sets concatenated to it in the LNKLST. (See Initialization and Tuning for more information on the LNKLST member in SYS1. PARMLIB.)

  BLDL searches the LNKLST lookaside directory by passing a parameter list to the subroutine CSVLLSCH. CSVLLSCH calls the LNKLST lookaside cross-memory search routine (CSVVLS01),

which searches the LNKLST lookaside directory for the member name and copies its directory entry into the passed area (the user's input parameter list or the BLDL work area, if a FIND request). Control is then returned to BLDL.

If the LNKLST lookaside directory is not available, the LNKLST PDS directories are searched.

- If SYS1.LINKLIB is not the referenced library, or if the LNKLST lookaside directory is not available, BLDL issues an EXCP to search the directory for a directory block with a key equal to or higher than the given member name. That directory block is read into virtual storage and searched for the matching member name.
- If the name is in the LNKLST lookaside directory, or the matching entry was found in a directory block, the directory entry is copied into the user's parameter list.
- If this is a FIND request, the relative address is copied into the DCBRELAD field in the DCB and control is passed to the POINT routine by issuing a BASSM instruction for supervisor key callers or a SYNCH macro instruction for user key callers. In either case, control is passed to a bootstrap routine in IGC018 (part of csect IECOSCR1) to allow a branch to the POINT routine in 24-bit addressing mode.
- If this is not a FIND request, BLDL obtains the next name in the parameter list to be matched, and continues the search.
- When the input parameter list has been completed, the routine returns control to the caller passing a return code in register 15.

Convert Relative-to-Full Address Routine—Entry Point: IECPCNVT: Conversion routine IECPCNVT accepts, in register 0, relative addresses of the form TTRN for direct-access devices and presents the corresponding full device addresses of the form MBBCCHHR at the location shown by register 2. This routine's external interface is documented in System-Data Administration.

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: Conversion 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 TTR0 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. This routine's external interface is documented in System-Data Administration.

Convert Record Number to Sector Value Routine—Entry Point: IECOSCR1: Conversion routine IECOSCR1 converts the record number for a fixed- or variable-length record data set to a sector value for use on an RPS device.

Note: For callers of this routine in 31-bit addressing mode, the +0 and +8 offsets into this routine have been changed to +16 and +20 respectively.

For fixed-length records, register 0 contains a data length in the two high-order bytes and a key length in the third byte.

The fourth byte contains the record number for which the sector value is desired.

For variable-length records, register 0 contains the number of key and data bytes already written in the two high-order bytes. The third byte contains a 1 (for keyed records) or a 0 (for nonkeyed records). The fourth byte contains the record number for which the sector value is desired.

For both types of records, registers 2, 14, and 15 contain the following:

Register 2 The high-order byte contains the UCB+19. The other 3 bytes contain the address at which the sector value is stored.

Register 14 The return address.

Register 15 The entry point address of this routine.

Upon completion, the sector value is stored at the designated address and registers 0, 9, 10, and 11 are modified.

Calculate Track Balance or Records per Track Routine—Entry Point IEC0SCR1+12: Within module IGC018, the conversion routine IEC0SCR1 calculates the new track balance or the number of records that can fit on a DASD track.

The routine input consists of:

- Device table address
- Record number
- Key length
- Data Length
- Track balance (optional)

Register 2 contains the address of this 12-byte parameter list.

The routine returns:

- In register 0, one of the following values:
  - The number of records that will fit on a track
  - The new track balance
  - The largest record that will fit on a track
- In register 14, the return address
- In register 15, the address of IEC0SCR1

Registers 9, 10, and 11 are work registers used by the routine.

The conversion routine is invoked via the TRKCALC macro. For information about the TRKCALC macro, see System-Data Administration.

SYNADAF AND SYNADRLS ROUTINES

See Diagram 0 for an illustration of the flow of processing through SYNADAF routines.

The SYNADAF routines pass control between modules by use of V-type address constants so as to maximize the chances of the next module being on the same page.

SYNAD Analysis and Format Routine IGC0006H: This routine is the SYNADAF SVC initial load module and the only load module for the SYNADRLS SVC. It gets storage for the register save area and the message buffer area and transfers control to the secondary load for error analysis. For SYNADRLS, it restores the save area pointers and frees gotten storage.

The routine operates as follows:

- It issues an ESTAE macro instruction to establish a task recovery routine (TRR) to intercept abnormal terminations while SYNADAF processing is in effect.
- It tests to determine whether it was entered for SYNADAF or SYNADRLS.
- If entered for SYNADAF:
  - It issues a GETMAIN macro instruction for a general register save area and a message buffer area from subpool 0, in the user's key.
  - It initializes the message buffer area.
  - It tests for a valid access method. If not valid, it issues an abend.
  - It loads the message CSECT.
  - It sets up the parameter list for transfer of control to secondary load routines for further analysis.
  - For BISAM or QISAM, it tests to determine if the DEB compatibility interface (CI) bit is set. If so, and the CI SYNADAF routine is provided, it transfers control to the SYNADAF routine via a SYNCH macro instruction. It returns to the caller when it again receives control.
  - If no CI SYNADAF routine is provided the routine returns to the caller.
  - If the DEB CI bit is not on, it branches to IGC0206H for BISAM and to IGC0306H for QISAM.
  - It branches to IGC0406H for BTAM, QTAM, or GAM.
  - It branches to IGC0506H for EXCP.
  - It branches to IGC0906H for BPAM or BDAM.
  - For BSAM or QSAM, it tests to determine if the DCB CI bit is on. If so, and the CI SYNADAF routine is provided, it branches to the SYNADAF routine via a SYNCH macro instruction. It returns to the caller when it again receives control.
  - If the SYNADAF routine is not provided, the routine returns to the caller.
  - If the DCB CI bit is not on, it branches to IGC0906H for BSAM or QSAM.
- If entered for SYNADRLS:
  - It restores the caller's save area pointer.
  - It releases the storage gotten for the register save area and the message buffer area.
  - It returns to the caller.

SYNADAF for BSAM, QSAM, BDAM, EXCP, and BPAM IGC0106H: This routine continues the error analysis for EXCP, BDAM, BPAM, BSAM, and QSAM and formats the message buffer. It receives control from IGC0506H for EXCP and from IGC0906H for the access methods.

The routine operates as follows:

- It tests to determine if the data set was opened.
- If the data set was opened:
  - It converts the DCB block count for tape and the IOB last seek address for disk storage into printable form.
  - It checks the ECB post codes.
  - If there is a permanent I/O error, it finds the IOBCSW for a unit check condition.
  - If there is a unit check, it transfers control to IGC0806H for the 3203 or 3211 Printers and 3800 Printing Subsystem and to IGC0706H for all other devices.
  - If there is no unit check, it deletes the message CSECT and returns to the caller.
- If the data set was not opened or if there was no permanent I/O error:
  - It examines the post code and formats the message accordingly.
  - It deletes the message CSECT and returns to the caller.

SYNADAF Routine for BDAM and BISAM IGC0206H: This routine completes the formatting of the message buffer for BDAM and BISAM.

It operates as follows:

- For BDAM:
  - It formats the DDNAME.
  - It searches the completion codes of the DECB and stores the related message.
  - If there is an IOB, it translates the Seek address into printable format. Else, it sets the Seek address field of the message buffer to zeros.
- For BISAM:
  - It searches the completion codes of the DECB and stores the related message.
  - It formats the device type and DDNAME and stores them in the message buffer.
- It deletes the message CSECT when formatting for BDAM or BISAM is complete.
- It returns to the caller.

SYNADAF for QISAM IGC0306H: This routine analyzes status and sense bytes and formats the condition portion of the message buffer, for QISAM.

The routine operates as follows:

- It formats the operation type.
- It tests the ECB post code.

- If the I/O event is not completed normally, it tests for an extent violation or a permanent I/O error and stores the corresponding error message.
- It analyzes the exceptional condition code and stores the error message.
- It formats the device type, unit ID, Seek address, and DDNAME.
- If there is no pointer to the DCB, it deletes the message CSECT and returns to the caller with a return code of 8 in register 15.
- Otherwise, it branches to IGC0406H for completion of the formatting.

SYNADAF ROUTINE FOR TCAM/QISAM IGC0406H: This routine continues the error analysis for GAM, BTAM, QTAM, QISAM, and TCAM.

It operates as follows:

- It receives control from IGC0006H for GAM, BTAM, or QTAM.
- It formats the access method type and stores "SYNAD ROUTINE NOT YET SUPPORTED" in the message buffer and returns to the user.
- It receives control from IGC0306H for QISAM.
- If the error is not a permanent I/O error, it searches the CSW status bytes and the IOB sense bytes and formats the related message text.
- It receives control from IGC0906H for TCAM.
- It stores the access method type in the message buffer.
- It checks to determine if the error is a work area overflow or an invalid destination. If neither is the cause of the error, the routine assumes a sequence error and stores the appropriate message text.
- It formats the operation type.
- It deletes the message CSECT and returns to the caller.

SYNADAF SVC IGC0506H: This routine formats the message buffer for EXCP.

The routine operates as follows:

- It stores the access method type in the message buffer.
- It obtains the operation code from the CCW and translates it into printable form.
- It validity-checks the UCB.
- If the UCB is not valid, it deletes the message CSECT and returns to the caller with a return code of 8 in register 0.
- It stores the unit ID in the message buffer.
- It branches to IGC0106H for further analysis.

SYNADAF SVC for OCR Load IGC0606H: This routine completes the formatting of the message buffer for OCR devices. IGC0606H receives control from IGC0906H.

It operates as follows:

- It translates the CCW operation code into printable form and stores it in the message buffer.
- It formats the DDNAME and stores it in the message buffer.
- It checks for the ECB completion code and stores it in the message buffer.
- It searches the IOBCSW status bytes and the IOB sense bytes and formats the appropriate text.
- The sense byte settings for the OCR are device-dependent so the routine increases the pointer to the appropriate message text and then stores the pointer in the message buffer.
- It deletes the message CSECT and returns to the caller.

SYNADAF Unit Check Analysis IGC0706H: This routine analyzes the ION sense bytes on a unit check condition for direct access, magnetic tape, and unit record devices, except for the 3203 or 3211 printers and 3800 Printing Subsystem. It receives control from IGC0106H and formats the message buffer and sets blanks in the work area.

It operates as follows:

- It scans the IOB sense bytes for error indications.
- It analyzes IOB sense bytes 0 and 1 and stores the corresponding error message in the message buffer.
- If there is a write-inhibit condition for a 3330 device, it stores the write-inhibit message text.
- If the unit record device type is a TCR (tape cartridge reader), the routine reanalyzes the sense bytes and stores the appropriate message text.
- It deletes the message CSECT and returns to the caller.

SYNADAF Unit Check Analysis IGC0806H: This routine analyzes the IOB sense bytes on unit check conditions for 3203, 3211, 3262 model 5, 4245, 4248 printers and 3800 Printing Subsystem.

IGC0806H receives control through a branch from IGC0106H on a unit check condition.

It operates as follows:

- It scans the IOB sense bytes for error indications.
- If the IOB sense bytes do not have the status indicator, it sets the message text to "unknown condition."
- It stores the appropriate message for the IOB sense bytes.
- It deletes the message CSECT and returns control to the caller.

SYNADAF Error Analysis IGC0906H: This module continues the generalized analysis of errors for BDAM, QSAM, BPAM, BSAM, and TCAM dummy data sets.

IGC0906H receives control through a branch from IGC0006H.

It operates as follows:

- For BDAM:

  - If the DECB does not contain an IOB address, the routine transfers control to IGC0206H for completion of the message buffer formatting.
  - If the DECB contains an IOB address, the routine formats the unit ID, device type, and operation type and branches to IGC0106H for further analysis.
- For BSAM, QSAM, and BPAM
  - It checks to determine if the device is an OCR. If so, it branches to IGC0606H.
  - It formats the operation type, unit ID, and device type.
  - It branches to IGC0106H for further analysis.
- For TCAM dummy data sets, it branches to IGC0406H.

SETPRT, SETDEV AND IMGLIB ROUTINES

When an SVC 81 (SETPRT or SETDEV) or SVC 105 (IMGLIB) instruction is issued, the executors associated with SETPRT, SETDEV, and IMGLIB receive control. Figure 33 on page 170 shows the flow of control among these executors.

When an SVC 81 instruction is issued for an IBM 3800 Printing Subsystem, IGC0008A is the initial executor to receive control. Executors IGG08110, IGG08111, IGG08112, IGG08113, IGG08114, and IGG08115 are then given control, depending on the contents of the SETPRT parameter list.

When an SVC 81 instruction is issued for a printer that is not an IBM 3800 Printing Subsystem, IGC0008A is the initial executor to receive control. Then, either executor IGG08105 for image table processing printers, or IGG08101 and IGG08102 in combination for nonimage table processing printers, may receive control from IGC0008A. The executors process the UCS request and perform image verification if required. Control may then be passed from IGC0008A, IGG08101, IGG08102, or IGG08105 to IGG08103 and IGG08104. Executors IGG08103 and IGG08104, respectively, locate the FCB image and load it into the printer's forms control buffer. Executor IGG08104 performs image verification or allows form alignment.

When a SETPRT is issued for a SYSOUT data set, executor IGC0008A gives control to executor IGG08117.

SETPRT is also issued during OPEN processing. When SETPRT is called by OPEN and the device is a printer other than a 3800 Printing Subsystem, SETPRT ensures that a UCS image and, for devices that have it, an FCB is or has been specified.

All messages issued by the SETPRT executors are in a message CSECT. The SETPRT executors must extract the text from the CSECT before issuing the message. If the user's key is greater than or equals 8, the SYNCH macro instruction is used for all WTO/WTORs for integrity reasons, because the message text is moved to the user's work area.

Two executors are associated with the SETDEV macro instruction. The SETPRT executor, IGC0008A, receives control when the SVC 81 instruction is issued. Executor IGG08108 receives control from executor IGC0008A to initialize the IBM 3890 Document Processor Control Unit.

Executor IGC0010E receives control when SVC 105 is issued to build or delete the DCB and DEB for a SYS1.IMAGELIB data set.

SETPRT Routine IGC0008A: The macro instruction SETPRT (set-printer) expands into an SVC 81 instruction that causes this routine to be loaded and to gain control. IGC0008A determines if the IBM 3890 Document Processor control unit is to be loaded and, if so, gives control to routine IGG08108. If the IBM 3800 Printing Subsystem is to be set up, control is given to IGG08110. If a SYSOUT data set is requested, control is given to routine IGG08117.

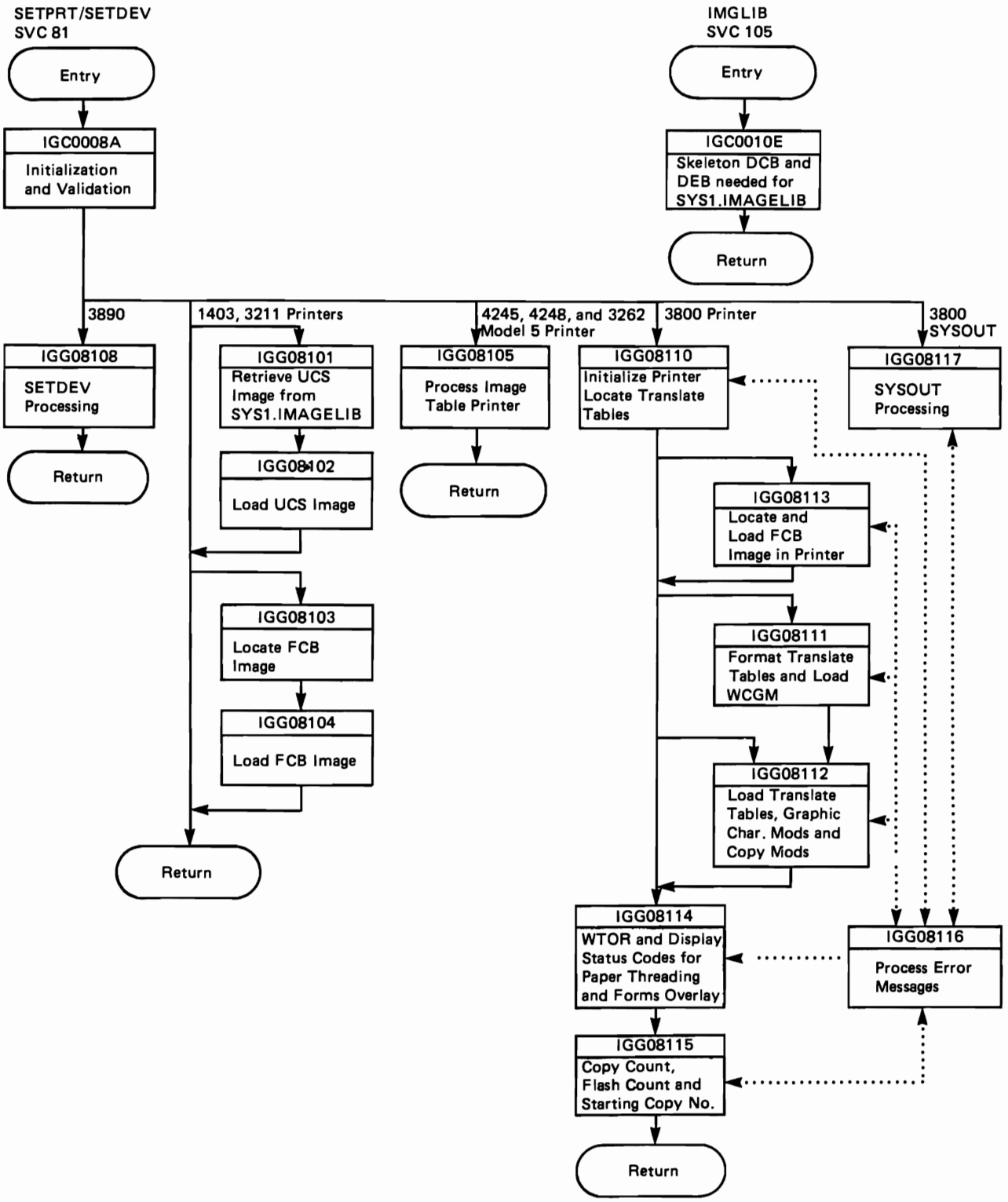


Figure 33. SETPRT Executor Selector

The SETPRT routine operates as follows:

- It issues an ESTAE macro instruction to establish a TRR, IGCT1081, to intercept abnormal terminations.
- It saves information in the SVRB extended save area for IGCT1081, in case the SETPRT ESTAE gets control.
- It contains a bootstrap routine that gets control from RTM; issues a LOAD macro instruction, followed by a DELETE macro instruction, for IGCT1081 (to get the entry point address of IGCT1081 in LPA); and branches to that module.
- It issues a LOAD macro instruction for message CSECT (IGGMSG01).
- It tests the DCB for a SYSOUT data set open for output and bypasses unit record processing.
- If the 3890 Document Processor control unit is to be loaded, it passes control to routine IGG08108.
- It uses the GETMAIN macro instruction to obtain two work areas.

  1. Key 5, subpool 230, for BLDL parameter list, a general work area.
  2. User key, subpool 230, another general work area. (See "SETPRT Work Area" in the "Data Areas" section.)

- It sets up various fields in the work areas for subsequent loads of SETPRT.
- When EXCP is specified in the DCB, or if OPEN is the caller of SETPRT, it builds an IOB in the user key work area.
- When QSAM is specified for the DCB, the routine causes all outstanding output requests to quiesce.
- If it is for a SYSOUT data set, control passes to routine IGG08117.
- It uses the EXCP macro instruction to execute block data check or reset block data check according to the specification in the SETPRT parameter list if any. If neither is specified, block data check is executed.
- If OPTCD=FOLD or UNFOLD is specified, an EXCP is issued to set the mode.
- It issues SVC 105 to open SYS1.IMAGELIB.
- If the device is a 3800 Printing Subsystem, control is given to executor IGG08110.
- When UCS image processing is required, it passes control to either executor IGG08101 for non-image table processing printers, or IGG08105 for image table processing printers.
- If FCB processing is required but UCS processing is not, routine IGG08103 is called.
- If no UCS or FCB processing is required, it frees the work area and returns to the caller with a return code in register 15. For a description of the return codes, see Data Administration: Macro Instruction Reference.

SETPRT Routine IGG08101: Routine IGG08101 is entered from routine IGC0008A when the specified UCS image is to be loaded from the SYS1.IMAGELIB.

The routine operates as follows:

- If OPEN is processing, the operator is requested to specify a UCS image name if none is specified and the currently loaded UCS image (if any) is not a default image.
- If OPEN is processing, no UCS image is specified, and the currently loaded UCS image, if any, is a default image, then the currently loaded image is made the requested UCS image in order to force reload of the UCS image and thereby ensure its integrity.
- It uses the BLDL macro instruction to locate the UCS image in the SYS1.IMAGELIB.
- If the UCS image is not in the library, the routine requests the operator to specify an alternate UCS image to be loaded.

  1. If the operator cancels the job step, it returns to the caller with a nonzero return code in register 15 (non-FCB printers). For FCB printers, it transfers control to IGG08103.

  2. If the operator replies "U", the currently loaded image is used.

- If an error occurs during BLDL processing, it returns control to the caller with a nonzero return code in register 15 (non-FCB printers). For FCB printers, it transfers control to IGG08103.
- When the UCS image is in the library, the routine uses the LOAD macro instruction to retrieve the UCS image from the library.
- Before returning to the caller, it issues a DELETE macro instructions for the message CSECT and for the UCS image, if it was loaded. It also frees the work areas and issues a CLOSE macro instruction for SYS1.IMAGELIB.
- The routine passes control to routine IGG08102 to load the retrieved UCS image into the UCS buffer.

SETPRT Routine IGG08102: Routine IGG08102 is entered from routine IGG08101 to load the UCS image into the UCS buffer and to print verification lines if required.

The routine operates as follows:

- It uses the EXCP macro instruction to load the UCS image into the UCS buffer.
- If an error occurs during UCS load, it returns control to the caller, with a nonzero return code for non-FCB printers. For FCB printers, it transfers control to IGG08103.
- Before returning to the caller when an error condition exists, it issues DELETE macro instructions for the message CSECT and for the UCS image, if it was loaded. It also frees the work areas and issues a CLOSE macro instruction for SYS1.IMAGELIB.
- When verification of the image is required, the routine uses the EXCP macro instruction to print the UCS image for verification.
- If an error persists during verification, it returns control to the caller with a nonzero return code in register 15 (non-FCB printers). For FCB printers, it transfers control to IGG08103.

- It gets the local lock and updates the UCB to reflect the new image. It then releases the local lock.

- It uses the DELETE macro instruction to release the UCS image. If no FCB processing is required, FREEMAIN macro instructions release the work areas, a DELETE macro instruction releases the message CSECT, and control is returned to the caller. IGC0008A, IGG08101, IGG08102, or IGG08105.

  The routine operates as follows:

  - If the UCS return code indicates a possible lost data condition or, if OPEN is processing and an error occurred during UCS processing, no FCB processing is performed.

  - It checks the DCB exit list to see whether the specified FCB image is defined in the problem program.

  - It uses the BLDL macro instruction to locate the FCB image in SYS1.IMAGELIB if the image is not specified in an exit list.

  - For the IBM 4248 Printer, the executor attempts to locate the FCB using the prefix FCB4 (a 4248 format FCB). If that FCB cannot be located, an attempt is made to locate the FCB using the prefix FCB2 (a 3211 format FCB).

  - If an error occurs during BLDL processing, it is remembered and control is given to IGG08104 with an indicator set that no FCB is to be loaded.

  - If the image is not found in the library, the routine requests the operator to specify an alternate FCB image.

    — If the operator cancels the job step, it is remembered and control is given to IGG08104 with an indicator set that no FCB is to be loaded.

    — If the operator replies "U", the currently loaded image is used.

    IGG08104 is called to load the image into the forms control buffer and print a verification, if requested.

SETPRT Routine IGG08104: Routine IGG08104 loads the FCB image into the forms control buffer and verifies the load and/or allows forms alignment. It is entered from IGG08103.

The routine operates as follows:

- If the FCB is found in an exit list or if OPEN is processing, a temporary copy of the requested FCB is created.

- If COPYP and/or PSPEED is specified in the SETPRT parameter list, if the FCB is found in an exit list, or if OPEN is processing, a temporary copy of the requested FCB is created. Any COPYP and PSPEED specifications are then copied into the temporary FCB.

- The FCB is loaded into the device.

- If an error occurs loading the FCB, a message to the operator is issued and a return code and a reason code are set.

- If VERIFY is specified, the image is printed for visual verification.
- If an I/O error occurs during verification (other than a possible lost data condition), control is returned to the caller. If a possible lost data condition is detected, the verification display is restarted up to 5 times before the condition is considered permanent. Control is returned to the caller with a nonzero return code in register 15.
- If ALIGN is specified, the operator is instructed to align the forms.
- It issues a DELETE macro instruction for the message CSECT and for the FCB image, if it was loaded from SYS1.IMAGELIB. It also issues a CLOSE macro instruction for SYS1.IMAGELIB and frees the work area gotten by IGC0008A.
- The routine always exits to the caller.

SETPRT Executor IGG08105: Executor IGG08105 receives control from IGC0008A and performs processing for image table printers.

The executor operates as follows:

- Processing continues if one of the following conditions is met:

  1. OPEN is processing and either a nondefault or no UCS image-id has been previously specified. In this case, the operator is requested to specify an image-id. The operator can reply with the id of an image to be used, and, optionally, fold and/or verify. The operator can also cancel the UCS request, or may reply "U" to indicate the currently mounted band is to be used.
  2. A UCS image is requested and:
     - It is not satisfied by the currently mounted band.
     - UCS verification is requested.
     - The fold/unfold mode differs from that previously set.

- The appropriate image table is located in SYS1.IMAGELIB and loaded into storage. The image table provides:
  - The correspondence between the UCS image-id (alias) and the printer band-id.
  - An indication as to whether the image is a default image.
  - A band description to be displayed as part of the verification display.
  - Optionally, the number and length of verification lines in the verification display.
- Whether initially requested, or specified by the operator, the name of the image to be used (alias) is searched for in the image table. If it is not found, the operator is asked to specify an alternate image-id. The operator can reply with the id of an image to be used, and, optionally, fold and/or verify. The operator can also cancel the UCS request or may reply "U" to indicate the currently mounted band is to be used.
- When the name of the image to be used is found in the image table, the operator is instructed to mount the appropriate band if it is not already mounted. If the printer is an IBM 4245, the image of the band to be mounted is also contained in the operator display.

After mounting the appropriate band, the operator indicates it is mounted by replying to the mount message with the same image-id the operator was instructed to mount. Alternatively, the operator can reply with the id of an image to be used, and, optionally, fold and/or verify. The operator can also cancel the UCS request or may reply "U" to indicate that the currently mounted band is to be used.

- When the operator indicates the correct band is mounted, the appropriate mode is set (fold or unfold).
- If the UCS image is to be verified, the following is performed:
  - If the printer is a 4248, the horizontal copy feature is deactivated.
  - The UCS verification display is produced. This display consists of a header line followed, optionally, by lines displaying the currently loaded image.
  - If the printer is a 4248, the horizontal copy feature is reactivated.
  - The operator is asked to verify the image by replying either:
    - VERIFY (the image is correct).
    - CANCEL (the image is not correct).
    - RETRY (the UCS verification display is given again).
  - If UCS processing is successful, the UCB UCS extension is updated to reflect the current image-id and mode.
- The image table is deleted.
- A freemain is issued for the work area.
- Executor IGG08103 is called.

SETDEV Routine IGG08108: IBM 3890 Document Processor routine. IGG08108 receives control from routine IGC0008A.

The routine operates as follows:

- It verifies the SETDEV parameter list.
- It loads the appropriate control unit.
- It exits to the problem program.

SETPRT Executor IGG08110 (For the 3800 Printing Subsystem only): Executor IGG08110 is entered from executor IGC0008A when the UCB device type indicates that SETPRT processing is being done for a 3800 Printing Subsystem.

The executor operates as follows:

- It builds a SETPRT path table according to the format and content of the SETPRT parameter list. The path table is used to control the sequence of execution for subsequent 3800 Printing Subsystem SETPRT executors.
- When printer initialization is requested, IGG08110 uses the EXCP macro to issue the channel command sequence that resets the controls for the 3800 Printing Subsystem.
- It issues the LOAD macro to load the requested character arrangement tables residing in SYS1.IMAGELIB (or at addresses provided by the caller).

- If an error is detected, it builds a message parameter list and calls executor IGG08116, then branches to executor IGG08115. Otherwise, it calls the next 3800 Printing Subsystem SETPRT executor specified in the SETPRT path table.

SETPRT Executor IGG08111 (For the 3800 Printing Subsystem only): Executor IGG08111 is entered when character arrangement tables are successfully obtained by IGG08110.

The executor operates as follows:

- It determines which character sets are needed for the character arrangement tables specified in the SETPRT parameter list.
- It reorders the character set positions to be loaded by referencing the current IDs as shown in the UCB. This minimizes the possibility of reloading previous character sets into the 3800 Printing Subsystem.
- It formats the translate tables, using the character arrangement tables.
- It uses the EXCP macro to load the 3800 Printing Subsystem's writable character generation modules (WCGMs) with the even-numbered hardware character sets.
- It issues a BLDL and a LOAD macro to read the odd-numbered library character sets into storage.
- It uses the EXCP macro to load the library character sets into the 3800 Printing Subsystem's WCGM storage with a load graphic character modification CCW.
- It issues a DELETE macro to free the storage used for the library character sets.
- If an error is detected, it builds a message parameter list and calls executor IGG08116, then branches to executor IGG08115. Otherwise, it calls the next 3800 Printing Subsystem SETPRT executor specified in the SETPRT path table.

SETPRT Executor IGG08112 (For the 3800 Printing Subsystem only): Executor IGG08112 is entered after successful processing by IGG08111, or when the SETPRT parameter list requires copy modification.

The executor operates as follows:

- It uses the EXCP macro to select the proper 3800 Printing Subsystem's translate table position(s) and to load the translate table(s) into the printer.
- When graphic modification modules are specified for the character arrangement tables, IGG08112 issues the LOAD macro to obtain the desired graphic modification modules from SYS1.IMAGELIB.
- It issues EXCP to load the graphic modification records into the 3800 Printing Subsystem.
- It issues a DELETE macro to free the storage used by the loaded graphic character modification modules.
- If a copy modification module (residing in SYS1.IMAGELIB) is requested, IGG08112 issues LOAD to retrieve it.
- It uses EXCP to load the 3800 Printing Subsystem with the copy modification record.
- It issues a DELETE macro to free the storage used by the loaded graphic character modification modules.

- If an error is detected, it builds a message parameter list and calls executor IGG08116, then branches to executor IGG08115. Otherwise, it calls the next 3800 Printing Subsystem SETPRT executor specified in the SETPRT path table.

SETPRT Executor IGG08113 (For the 3800 Printing Subsystem only): Executor IGG08113 is entered when forms control buffer image processing is required.

The executor operates as follows:

- It checks for the specified FCB image identifier using FCB entries in the DCB exit list.
- When the address of the FCB image is not passed to the caller and the FCB image cannot be located by the DCB exit list, the FCB image is obtained from the SYS1.IMAGELIB data set using the LOAD macro.
- It loads the specified FCB image into the 3800 Printing Subsystem.
- If FCB image verification is requested, IGG08113 formats and prints on the 3800 Printing Subsystem a map of the specified FCB image. A message, asking for visual verification, is sent to the operator.
- If an error is detected, it builds a message parameter list and calls executor IGG08116, then branches to executor IGG08115. Otherwise, it calls the next 3800 Printing Subsystem SETPRT executor specified in the SETPRT path table.
- If an error is detected, it calls executor IGG08115. Otherwise, it calls the next 3800 Printing Subsystem executor specified in the SETPRT path table.

SETPRT Executor IGG08114 (For the 3800 Printing Subsystem only): Executor IGG08114 is entered when the UCB extension indicates that the 3800 Printing Subsystem has the burster-trimmer-stacker feature installed, or when forms overlay processing is requested.

The executor operates as follows:

- It verifies that the paper is positioned properly, and informs the operator if paper repositioning is required.
- When the FLASH parameter is used, it requests the operator to install the requested forms overlay negative.
- It passes control to executor IGG08115.

SETPRT Executor IGG08115 (For the 3800 Printing Subsystem only): Executor IGG08115 is the last module executed for 3800 Printing Subsystem SETPRT processing.

The executor operates as follows:

- If no errors were detected by previous 3800 Printing Subsystem SETPRT executors, IGG08115 initializes the 3800 Printing Subsystem printer with: the starting copy number, the total copies to be printed, and the copies to contain a forms overlay image.
- It resets the translate table index in the 3800 Printing Subsystem to the first translate table loaded.
- It restores the caller's control blocks as required, deletes any loaded modules, and frees previously obtained virtual storage loaded or acquired by or for SETPRT.
- It exits to the issuer of the SVC 81.

SETPRT Executor IGG08116: Executor IGG08116 is the error message processing routine and is called by the other 3800 Printing Subsystem SETPRT executors when an error is detected.

The executor operates as follows:

- It checks the return code and, if an I/O error is indicated, it retrieves the opcode of the failing CCW and stores it into byte 0 of the reason code.
- It uses the message parameter list passed by the caller in the SETPRT work area, and formats the corresponding message in a work area buffer.
- If the message suppression bit in the SETPRT parameter list is off, it will do the following:

  1. If a SYSOUT error or a previous I/O error is indicated, a WTO to the programmer is issued.
  2. For all other messages, except for a BURST request error, an initialize printer CCW is issued to ensure a standard setup, and the error message is created on the 3800 Printing Subsystem. For a BURST request error, the setup is left unchanged, and the message is written with the current setup for informational purposes.
  3. If a paper jam or cancel key condition is detected, the corresponding message is formatted in the work area buffer, and the return and reason codes are updated. Because the printer is in a not-ready state, no message is written to the 3800 Printing Subsystem.

- If a message feedback area is provided by the user program, the message text is copied from the work area buffer into the user-specified area.
- It returns to the calling program through a BR 14 instruction.

SETPRT Executor IGG08117: Executor IGG08117 is entered when the DCB indicates that SYSOUT processing is requested.

The executor operates as follows:

- It copies the user's SETPRT parameter list into the key 5 storage in the work area.
- It uses the GETMAIN macro instruction to obtain two key 5 work areas for SYSOUT processing.

  - A work area for subsystem interface control blocks, JFCB and JFCBE control blocks, and for information saved to communicate with the system work area (SWA) manager routines.
  - A parameter list for spool file allocation routine.

- It calls the SWA manager to read a copy of the JFCB into the work area.
- If a JFCBE exists, it calls the SWA manager to read a copy of the JFCBE into the work area.
- It issues an ENQ macro (RET-HAVE) on the TIOT. This will serialize SETPRT processing with any concurrent open, close, allocation, or unallocation processing against the data set.
- It checks DSABOPCT in the DSAB to ensure that only one DCB is open for the data set.
- It calls the CLOSE subsystem interface to notify JES that a data set segment is finished.

- It updates local copies of the JFCB and JFCBE according to information in the SETPRT parameter list.
- It builds the spool file allocation parameter list and calls the scheduler spool file allocation routine (IEFAB4SF). This will provide JES with new setup requirements, segment the data set, and update the JFCB and JFCBE in the SWA.
- It calls the OPEN subsystem interface to notify JES that a new data set segment is to be created.
- The executor frees up the SETPRT resources before returning to caller as follows:
  - It issues a DEQ macro for the TIOT.
  - It deletes the message CSECT (IGGMSG01).
  - It restores the IOB.
  - It saves return and reason codes in the SVRB.
  - It issues the FREEMAIN macro to:
    - Free the user key SETPRT work area
    - Free the spool file allocation parameter list area
    - Free the SYSOUT work area
    - Free the key 5 SETPRT work area
- It places the return and reason codes in registers 15 and 0, and returns to the caller.
- If an error is detected, it builds a message parameter list and calls executor IGG08116, then cleans up any existing resources before returning to the caller.

IMGLIB SVC Routine IGC0010E: The IMGLIB routine IGC0010E builds a skeleton DCB and DEB for the SYS1.IMAGELIB data set or deletes the DCB and DEB for the SYS1.IMAGELIB data set, depending on the parameter passed to it in register 1. The routine is entered from the SVC 105 instruction.

The IMGLIB macro is issued by OPEN executors and by SETPRT routines and can be issued by users. The routine operates as follows:

- It issues an ESTAE macro instruction to establish a TRR, IGCT010E, to intercept abnormal terminations.
- It makes a test to determine whether the control blocks for IMAGELIB need to be built or deleted. If register 1 contains 0's, a DCB and DEB are built.
- It uses a GETMAIN macro instruction to obtain a work area and then uses a LOCATE macro instruction to determine where the IMAGELIB volume is residing.
- It takes the address of the UCB table from the CVT and searches for the corresponding UCB.
- It uses the OBTAIN macro instruction to read in the format-1 DSCB and uses the information read and the UCB address to construct a skeleton DCB and DEB for the SYS1.IMAGELIB volume. The format-1 DSCB describes up to three extents. The SYS1.IMAGELIB data set can reside on up to 16 extents on a permanently resident volume.
- If there are more than three extents on SYS1.IMAGELIB, the format-3 DSCB seek address is obtained from the format-1 DSCB. It uses the OBTAIN macro to read in the format-3 DSCB

and uses the information read and the UCB address to construct additional DEB extent descriptions.

TASK RECOVERY ROUTINES

Task recovery routines are designed to minimize overhead in the execution of the following SVC routines.

SVC 18 BLDL or FIND
SVC 21 STOW
SVC 24 DEVTYPE
SVC 25 Track Balance, Track Overflow Erase
SVC 68 SYNADAF/SYNADRLS
SVC 69 BSP
SVC 81 SETPRT
SVC 105 IMGLIB

Explicit validity checking is not done in SVC routines. Instead, the following precautions are taken to ensure system integrity:

- Perform all read and write access to user-owned storage in the key of the caller.
- Issue an EXCP macro instruction on caller-owned control blocks in the key of the caller.
- Issue a SYNCH macro instruction to reach processing routines whose addresses are obtained from the caller's DCB.

SVC routines do not use storage that can be altered by a problem program for sensitive data that specifies the location of protected control blocks or the location to which control will be passed. Examples are register save areas and XCTL lists.

A program check occurs when a user error or a deliberate action threatens to impair system integrity. To avoid the situation in which the user would have to relate a program check in an unfamiliar system routine with an error in a problem program, each SVC routine has a task recovery routine on its RB level. On entry, each SVC routine issues an ESTAE macro instruction to establish the task recovery routine that will intercept abnormal terminations.

When a program check occurs, the task recovery routine is given control by RTM and performs explicit validity checking on the input to the SVC routine to determine if a user input error occurred. The task recovery routine can thus provide the caller with an error description in terms of the caller's input to the SVC routine.

Alternatively, the task recovery routine can determine that the abend was not caused by a user error, but was the result of an environmental situation or a system error. In the latter case, the error descriptive information can be directed to system data sets, rather than to the problem program user.

Task Recovery Routine IGCT0018 (SVC 18, BLDL or FIND): Module IGCT0018 handles abends arising from the issuance of an SVC 18.

It operates as follows:

- It analyzes the type of error and either passes it to the abend routine unchanged or changes it to its own abend and passes it on, or it uses a RETRY routine to change it into a user error abend code.
- If a system error is detected, this routine writes a record to SYS1.LOGREC and, if no lower level recovery routine has done so, takes a dump to SYS1.DUMP.
- It cleans up any resources that BLDL acquired.

Task Recovery Routine Module IGCT0021 (SVC 21, STOW): Module IGCT0021 analyzes abnormal terminations that result from issuing the STOW SVC.

The module operates as follows:

- It determines whether user input or system error caused the termination.
- It routes information about the abend to the appropriate place.
- For a user input error, it sends a dump of virtual storage to the SYSABEND or the SYSUDUMP file.
- For a system error, it notifies the problem programmer via a WTP message and a system completion code that an error occurred that terminated the task.
- The corresponding diagnostic information is written to SYS1.DUMP and SYS1.LOGREC.
- It performs any necessary cleanup.
- It records its actions when an error is detected and analyzed on a lower level, or when a machine check occurred and has been processed by the machine check handler.

Task Recovery Routine IGCT002D (SVC 24, DEVTYPE): Module IGCT002D analyzes abnormal terminations that occur as a result of issuing the DEVTYPE SVC.

The module operates as follows:

- It determines whether user input or system error caused the termination.
- It routes information about the abend to the appropriate place.
- For a user input error, it sends a dump of virtual storage to the SYSABEND or the SYSUDUMP file.
- For a system error, it notifies the problem programmer via a WTP message and a system completion code that an error occurred that terminated the task.
- The corresponding diagnostic information is written to SYS1.DUMP and SYS1.LOGREC.
- It performs any necessary cleanup.
- It records its actions when an error is detected and analyzed on a lower level, or when a machine check occurred and has been processed by the machine check handler.

Task Recovery Routine IGCT002E (SCV 25, Track Balance, Track Overflow Erase): Module IGCT002E analyzes abnormal terminations that occur as a result of issuing SVC 25. The module operates as follows:

- It determines whether user input or system error caused the termination.
- It routes information about the abend to the appropriate place.
- For a user input error, it sends a dump of virtual storage to the SYSABEND or the SYSUDUMP file.
- For a system error, it notifies the problem programmer via a WTP message and a system completion code that an error occurred that terminated the task.

- The corresponding diagnostic information is written to SYS1.DUMP and SYS1.LOGREC.
- It performs any necessary cleanup.
- It records its actions when an error is detected and analyzed on a lower level, or when a machine check occurred and has been processed by the machine check handler.

Task Recovery Routine IGCT006H (SVC 68, SYNADAF/SYNADRLS): Module IGCT006H determines the type of error that occurred during a SYNADAF or SYNADRLS SVC processing.

The module operates as follows:

- It requests a retry if SYNADAF attempts to access a control block at an invalid or a fetch-protected location.
- All other error conditions are percolated after the SYNADAF save/message area is freed. If this routine detects a system error, it issues an SDUMP before freeing the area.
- It consists of the following subroutines:

  Environment Error

  If entered from SYNADAF, it goes to the cleanup routine for normal processing.

  Otherwise, it tests for an error during FREEMAIN macro instruction processing in SYNADRLS.

  If not, the routine next tests for a user error.

  If the error occurred during FREEMAIN processing, it tests for an invalid FREEMAIN.

  If so, a SETRP macro instruction is issued to effect a retry.

  The retry routine restores register 13 in the SVRB to the address of the attempted FREEMAIN area and returns to the user with return code 8 in register 0.

  User Error

  Entered if (1) a program check occurred at the SYNADAF level and no subordinate ESTAE macro instructions were issued or (2) the environment error routine test indicated no FREEMAIN error.

  It tests for a protection exception, request for an invalid page, or segment exception.

  If so, it tests to determine if the retry address indicates that the control block address supplied by the user is invalid. If so, abend is invoked.

  Otherwise, the recovery routine attempts to continue SYNADAF processing. If the retry register is not set to zeros, the retry is performed by zeroing the retry register and issuing a SETRP macro instruction to retry at the address previously in the retry register.

  If any of the user routine tests fail, the error is treated as a SYNADAF failure, it is assumed to be a system error, and an SDUMP is invoked.

  SDUMP

  Functions performed are the same as for the user routine, with the following exceptions:

1. The abend code for the SDUMP header is taken from SDWACMPC.

2. The following additional information is logged: ",CODE=XXX,RC=NN."

3. The WTP message states, "IEC906I POSSIBLE SYSTEM ERROR DETECTED BY SYNADAF. SVC DUMP TRIED, RC=NN."

4. The abend code is not modified.

Cleanup

Records the following information in the LOGREC variable area: FLAG, RETRY, EP, USER15, USER0, and USER1.

SDWAURAL is inspected to avoid overlaying information already in the variable area, and is updated.

It deletes the message CSECT, if loaded, and frees the save/message area.

Before issuing the FREEMAIN macro instruction, it restores user register 13 from the HSA word of the save area being freed.

It then percolates the abend.

Task Recovery Routine IGCT0069 (SVC 69, BSP): Module IGCT0069 analyzes abnormal terminations that occur as a result of issuing the BSP SVC.

The module operates as follows:

- It determines whether user input or system error caused the termination.
- It routes information about the abend to the appropriate place.
- For a user input error it sends a dump of virtual storage to the SYSABEND or the SYSUDUMP file.
- For a system error, it notifies the problem programmer via a WTP message and a system completion code that an error occurred that terminated the task.
- The corresponding diagnostic information is written to SYS1.DUMP and SYS1.LOGREC.
- It performs any necessary cleanup.
- It records its actions when an error is detected and analyzed on a lower level, or when a machine check occurred and has been processed by the machine check handler.

Task Recovery Routine IGCT1081 (SVC 81, SETPRT): Module IGCT1081 receives control when the RTM (recovery termination manager) detects an abnormal termination during operation of SVC 81.

The module operates as follows:

- It issues an IGGSTART macro instruction to determine what type of processing should be done and to give control for error analysis to the appropriate routine.
- It contains the following routines:

  Resource Cleanup Routine

  Releases or restores resources acquired or altered by the SETPRT routines.

Establishes a second-level ESTAE in case a program check occurs during cleanup.

Exit Routine

Determines the correct way to exit from this module.

1. Percolate: If entered for resources cleanup, to take an SDUMP, or detection of a system error.

2. Retry to abend: If an invalid user control block was found.

   Uses the SETRP macro instruction to exit.

User Control Block Error Analysis Routine

Analyzes user control blocks, using the supervisor validity check routine and the DEBCHK routine. Issues a GETMAIN macro instruction for a GTF buffer for tracing control blocks.

If an invalid control block is detected, gives control to the GTF routine; otherwise, a system error is assumed.

SDUMP Routine

Issues an SDUMP SVC.

Builds a WTP (write-to-programmer) message if it determines that a system error occurred.

GTRACE Routine

Moves selected user control blocks to the GTF buffer.

Issues a GTRACE macro instruction.

Frees the GTRACE buffer.

Write-to-Programmer Routine

Builds a WTP message, if requested.

Issues a WTP macro instruction.

Retry Routine

Issues an ESTAE 0 (to disestablish the first level ESTAE), restores the user's registers, and issues an abend macro instruction.

Second-Level ESTAE Return Routine

Provides a return address for the second-level ESTAE.

Issues an ESTAE 0 to disestablish the ESTAE issued by the cleanup routine and then returns to the cleanup routine.

- This module also has a second-level ESTAE, which is entered if a program check occurs in the first-level ESTAE or if a CALL RTM is issued.

  For program checks, it issues a SETRP macro instruction to retry back to the second-level ESTAE return routine in the first level ESTAE. For a CALL RTM, it takes a system dump, issues a WTP macro instruction, and percolates to the next-level ESTAE.

Task Recovery Routine IGCT010E (SVC 105, IMGLIB): Module IGCT010E determines the type of error that occurred during, or that is related to, the IMGLIB SVC.

The module operates as follows:

- It determines if the error type is environmental, user, or system.
- It decides whether to return to the user, with a completion code, or to continue the abend.

GTRACE Record Format Module AMDUSRFE: Module AMDUSRFE formats all BSAM, QSAM, BPAM, and BDAM trace records created by the GTRACE macro instruction and causes them to be printed by the EDIT function of the AMDPRDMP service aid. It receives control from the EDIT function and can be used by both ABDUMP/SNAP and EDIT to format the user trace records.

The module operates as follows:

- When the module receives control, register 1 contains the address of a parameter list. It uses the parameter list to find the record to be processed, determine how to process it, and decide where to put the processed record.
- It saves the registers in the save area provided by EDIT.
- It moves the generalized block heading, "BSAM/QSAM/BPAM/BDAM TRACE RECORD DDNAME XXXXXXXX ABEND CODE XXX RETURN CODE XX TIME HH.MM.SS.HT" into the output buffer area provided by EDIT.
- It returns to EDIT with a return code of 0, which requests EDIT to print the output buffer area, clear the output buffer area, and return to the format module.
- The block heading data is derived as follows:

  DDNAME and RETURN CODE are taken from the data portion of the input trace record. The RETURN CODE is converted from binary code into printable form.

  ABEND CODE is the translation of EID (EVENT ID) from a 2-byte code into a 3-byte completion code.

  It checks the GTF option word (byte 4, bit 7) to determine if the TIME field is present. If the bit is off, the TIME field in the block heading is left blank. TIME is the local time that the record was put into the trace buffer. It is taken from the TIMESTAMP field and is converted into printable form.

- It provides a record heading, followed by data lines, for each type of logical record traced.
- It uses the ID of the traced record to determine what logical record is in the input buffer and then moves in the appropriate record heading. For example, if the ID indicates a DCB record (ID=130), the record heading moved into the output buffer is "DATA CONTROL BLOCK AT LOCATION XXXXXXXX."
- It again passes control to EDIT with a return code of 0.
- Upon return from EDIT, it sets up the data lines to be printed. Each line is moved into a work area for unpacking and translation into printable format.
- It moves the translated data into the output buffer in sets of 8 bytes, followed by 2 blank bytes.
- It tests for an end of data condition as the buffer is being filled.

When the end-of-data is reached, it restores the registers and exits to EDIT with a return code of 4, which requests EDIT to print the contents of the output buffer and obtain the next input trace record.

If it is not an end of data, it passes control to EDIT with a return code of 0 to cause the output buffer to be printed and control returned to the formatting module. This continues until all records in the input area are printed.

DIAGRAM A: SEQUENTIAL ACCESS METHODS—OVERVIEW

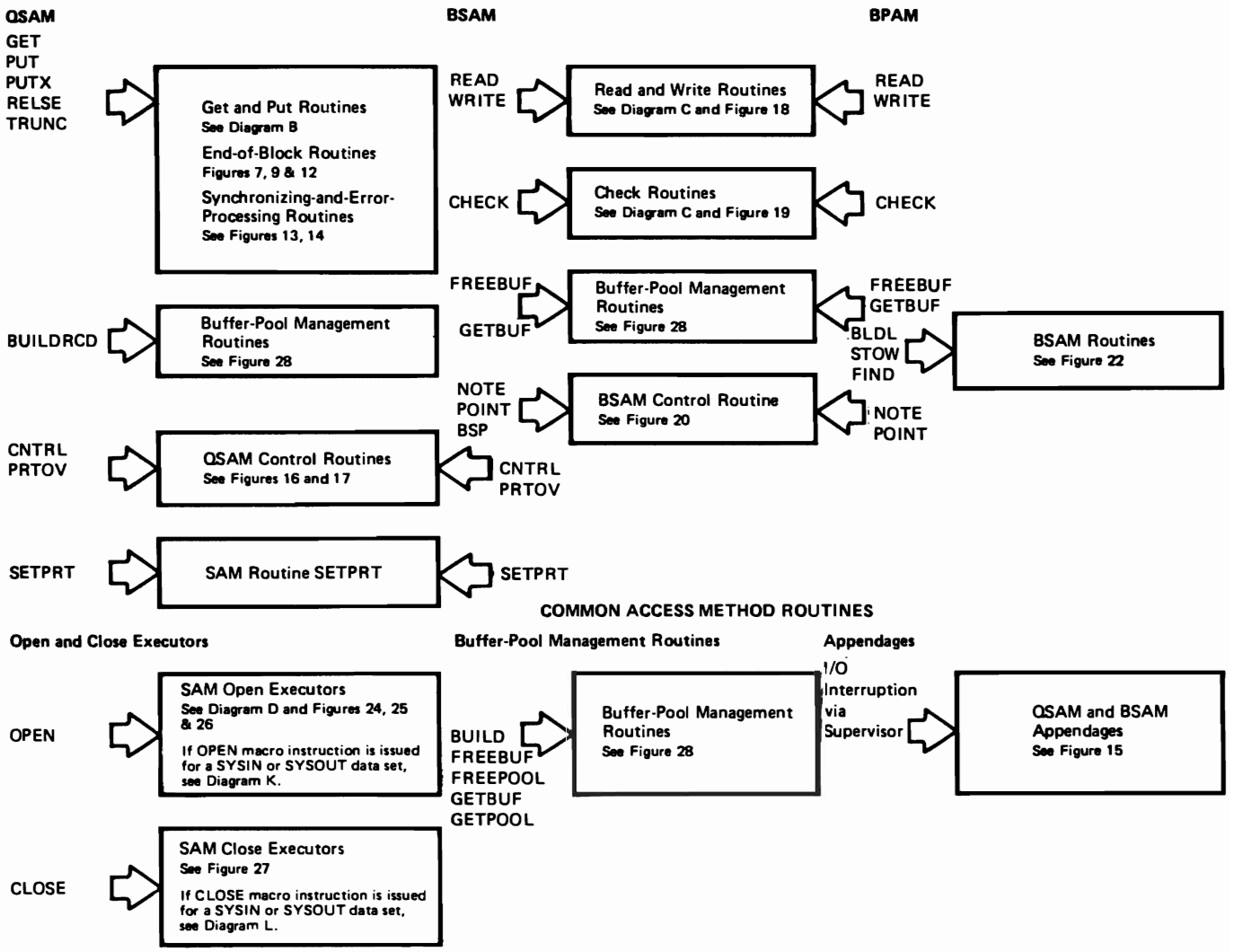


DIAGRAM B: QSAM GET AND PUT ROUTINES

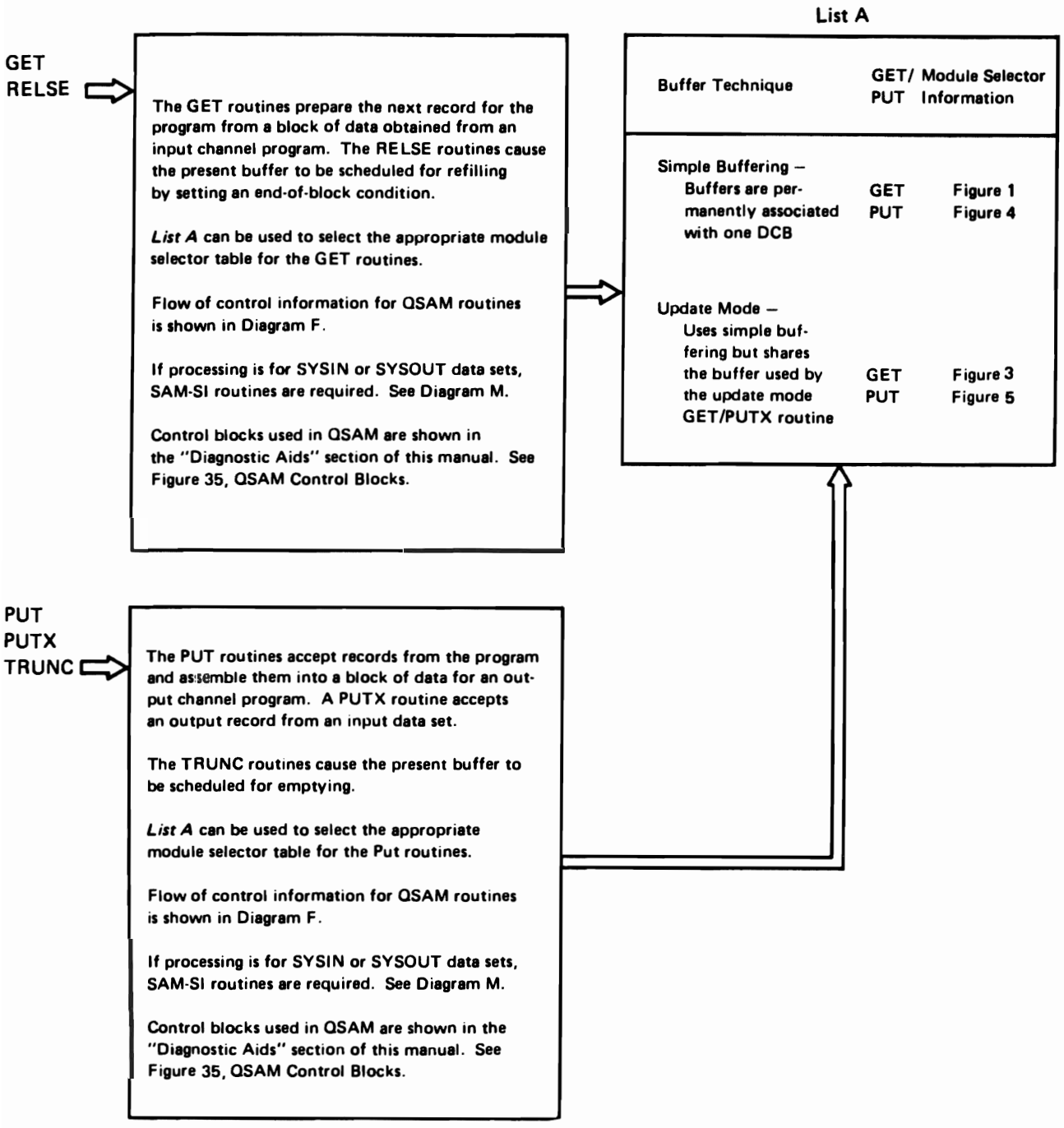


DIAGRAM C: BSAM/BPAM READ/WRITE AND CHECK ROUTINES

READ/WRITE ⇨

A READ or WRITE routine completes some of the entries in the channel program from parameters in the data event control block (DECB).

The READ/WRITE modules are listed in Figure I6.

For flow of control information for BSAM/BPAM routines, see Figure 20 and Diagram G.

If processing is for SYSIN or SYSOUT data sets, SAM-SI routines are required. See Diagram M.

Control blocks used in BSAM are shown in the "Diagnostic Aids" section. See Figure 36, BSAM Control Blocks.

CHECK ⇨

The DECB is examined by a CHECK routine to determine the status of the channel program.

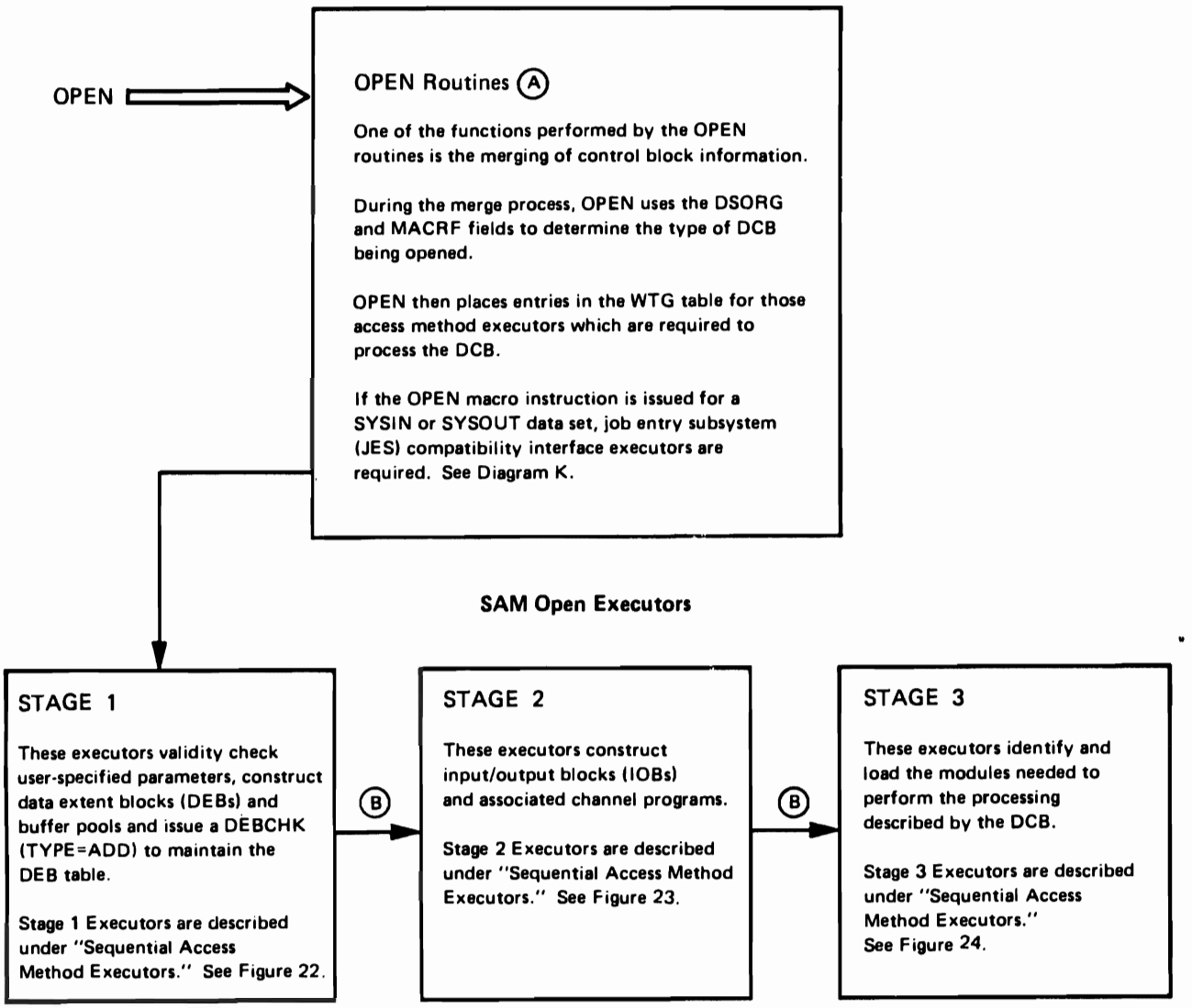
The CHECK modules are listed in Figure I7.

For flow of control information for BSAM/BPAM routines, see Figure 20 and Diagram G.

If processing is for SYSIN or SYSOUT data sets, SAM-SI routines are required. See Diagram M.

Control blocks used in BSAM are shown in the "Diagnostic Aids" section. See Figure 36, BSAM Control Blocks

DIAGRAM D: SEQUENTIAL ACCESS METHOD OPEN EXECUTORS



(A)	Open routines are described in ***OPEN/CLOSE/EOV Logic***. For information on the WTG and XCTL tables, see the "Access Method Determination" section of the manual.
(B)	Diagram E shows the flow of control among the three stages of OPEN Executors.

DIAGRAM E: STAGE 1—SAM FLOW OF CONTROL FOR OPEN EXECUTORS

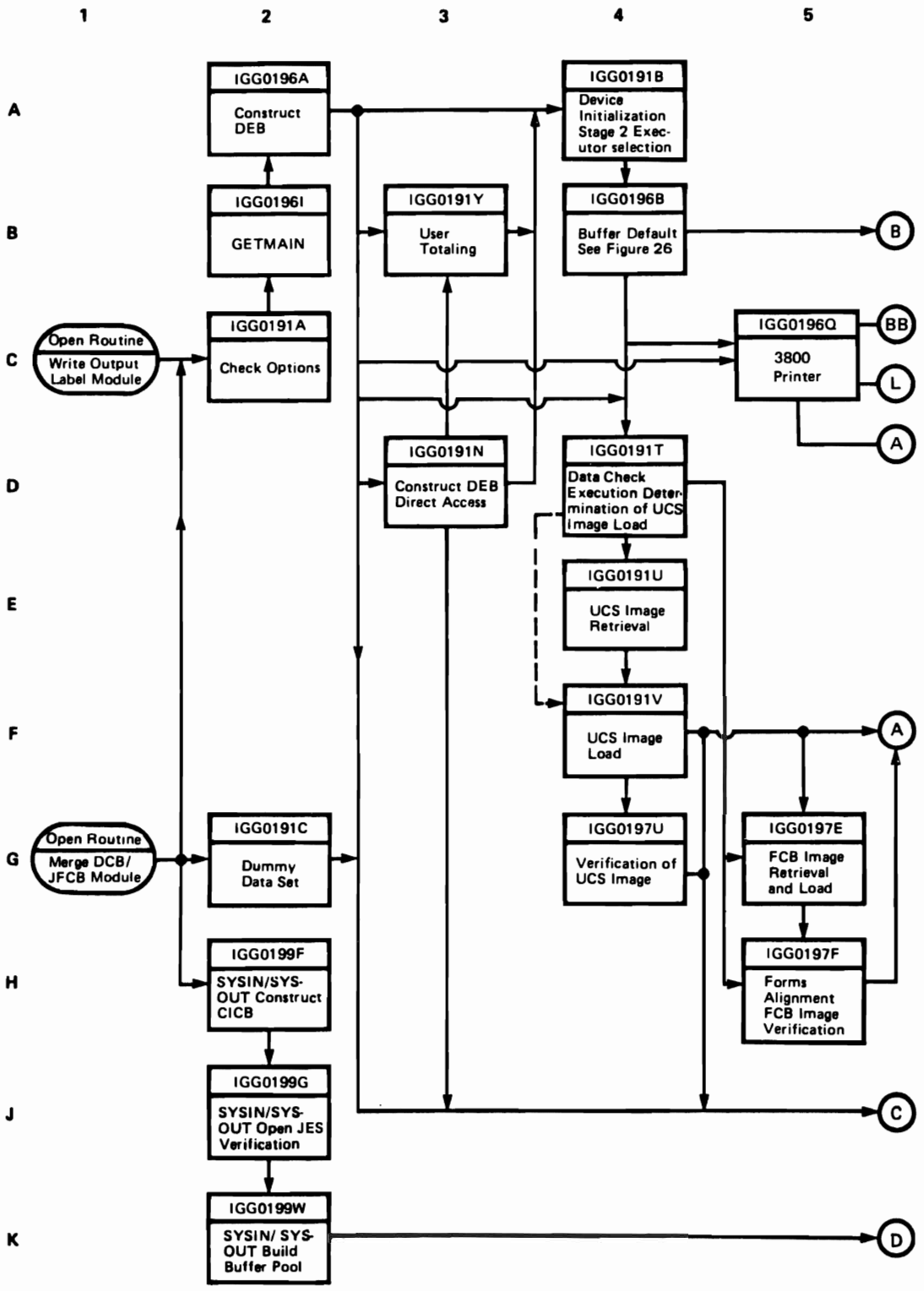


DIAGRAM E: STAGE 2—SAM FLOW OF CONTROL FOR OPEN EXECUTORS

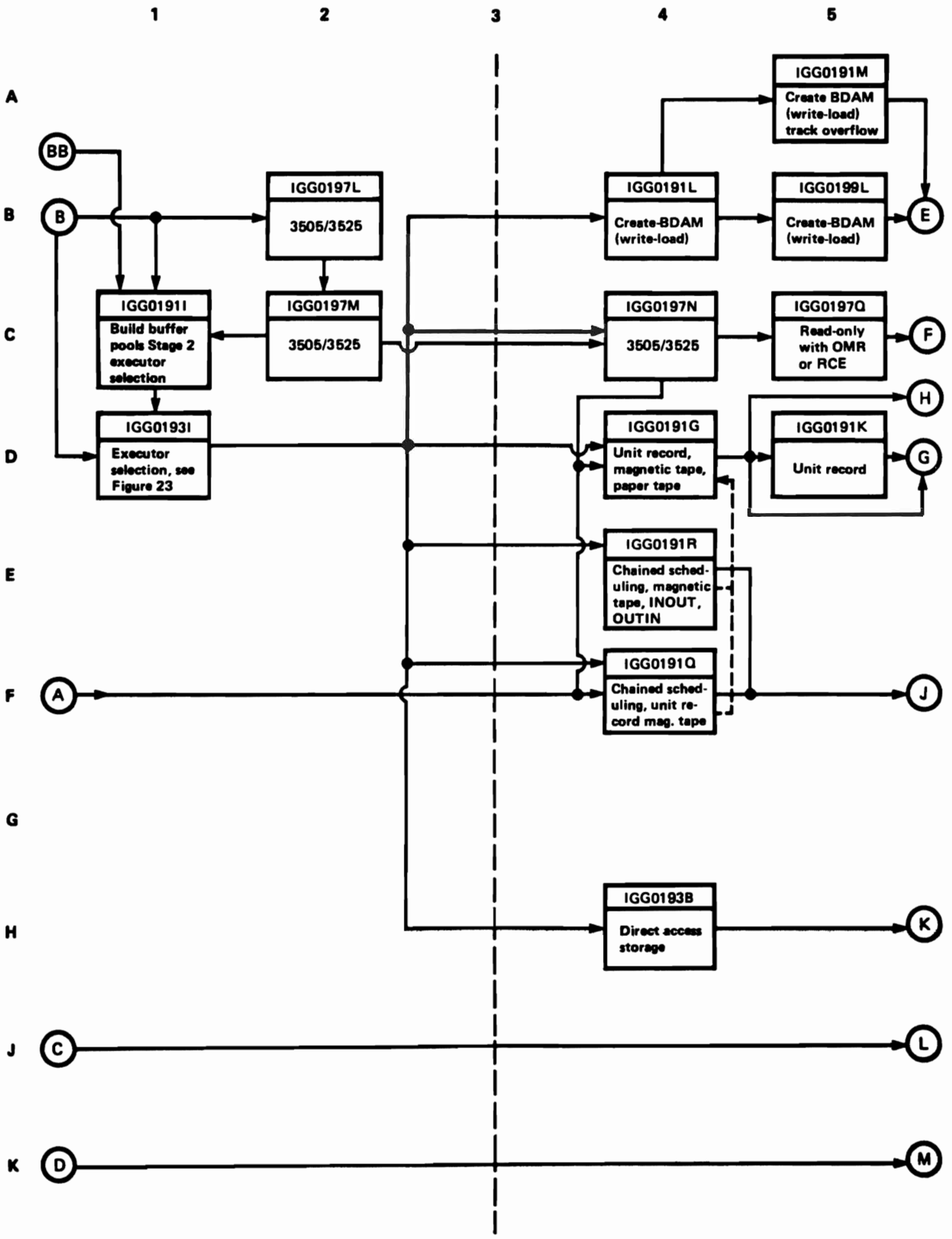


DIAGRAM E: STAGE 3—SAM FLOW OF CONTROL FOR OPEN EXECUTORS

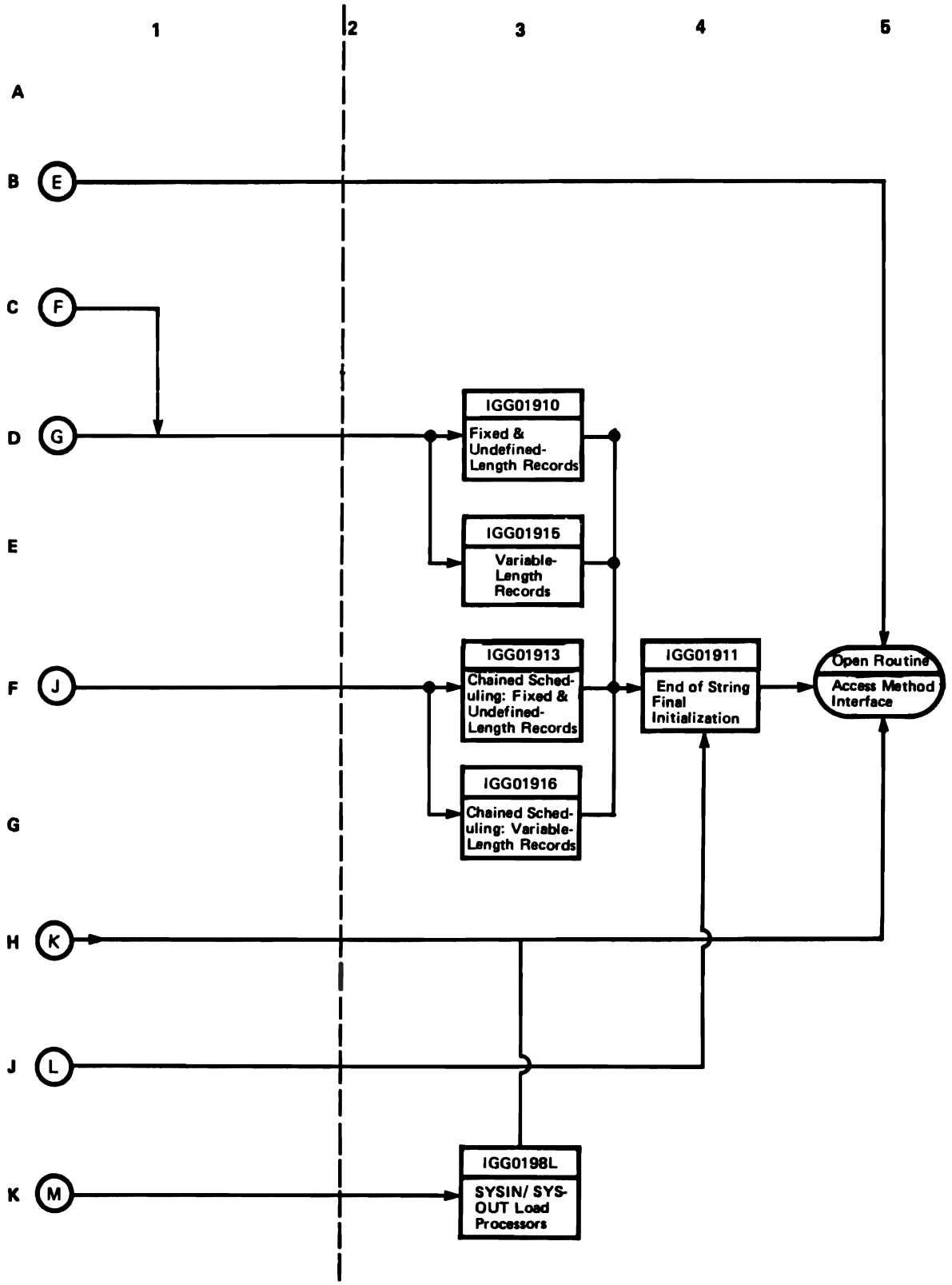
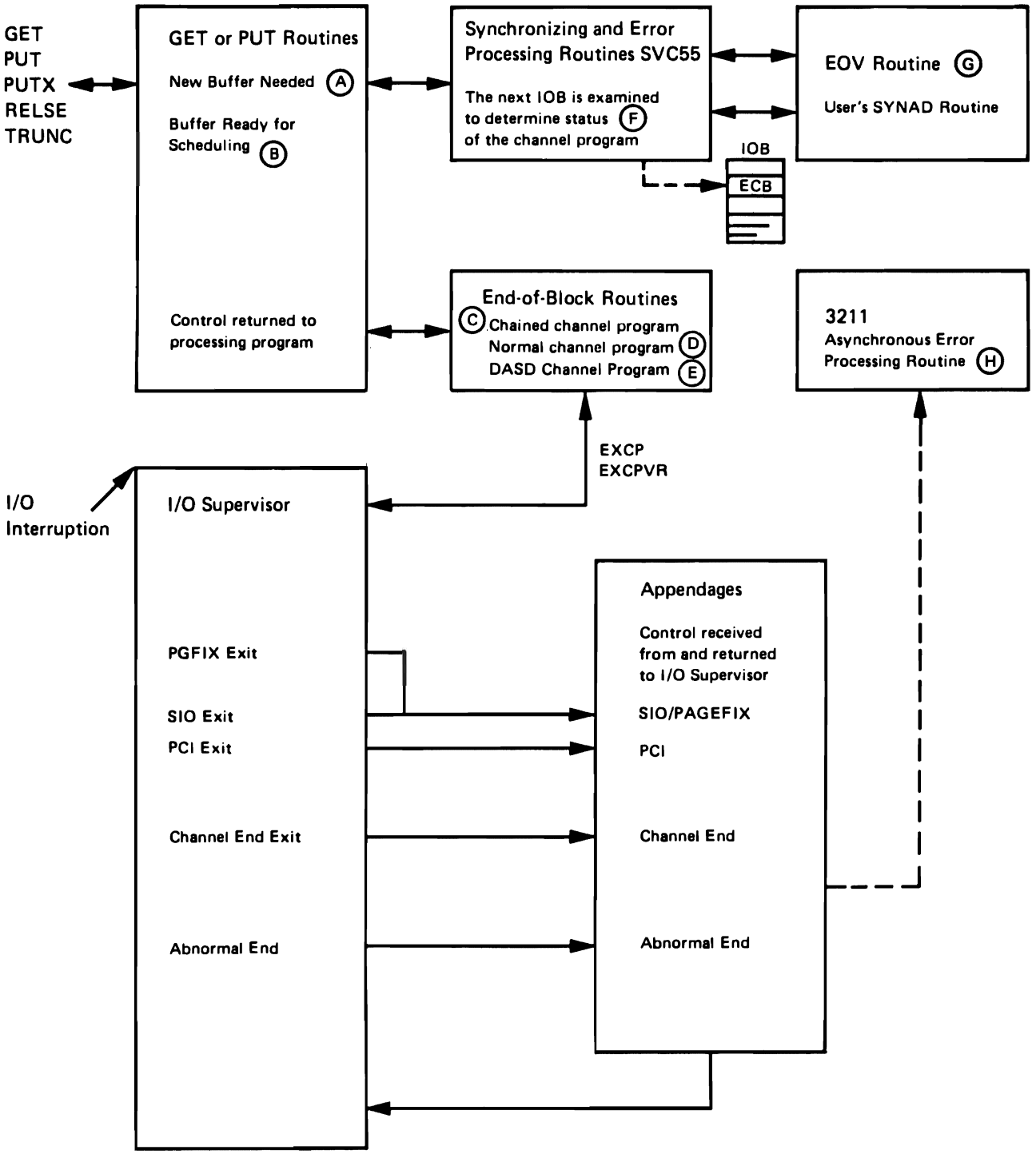


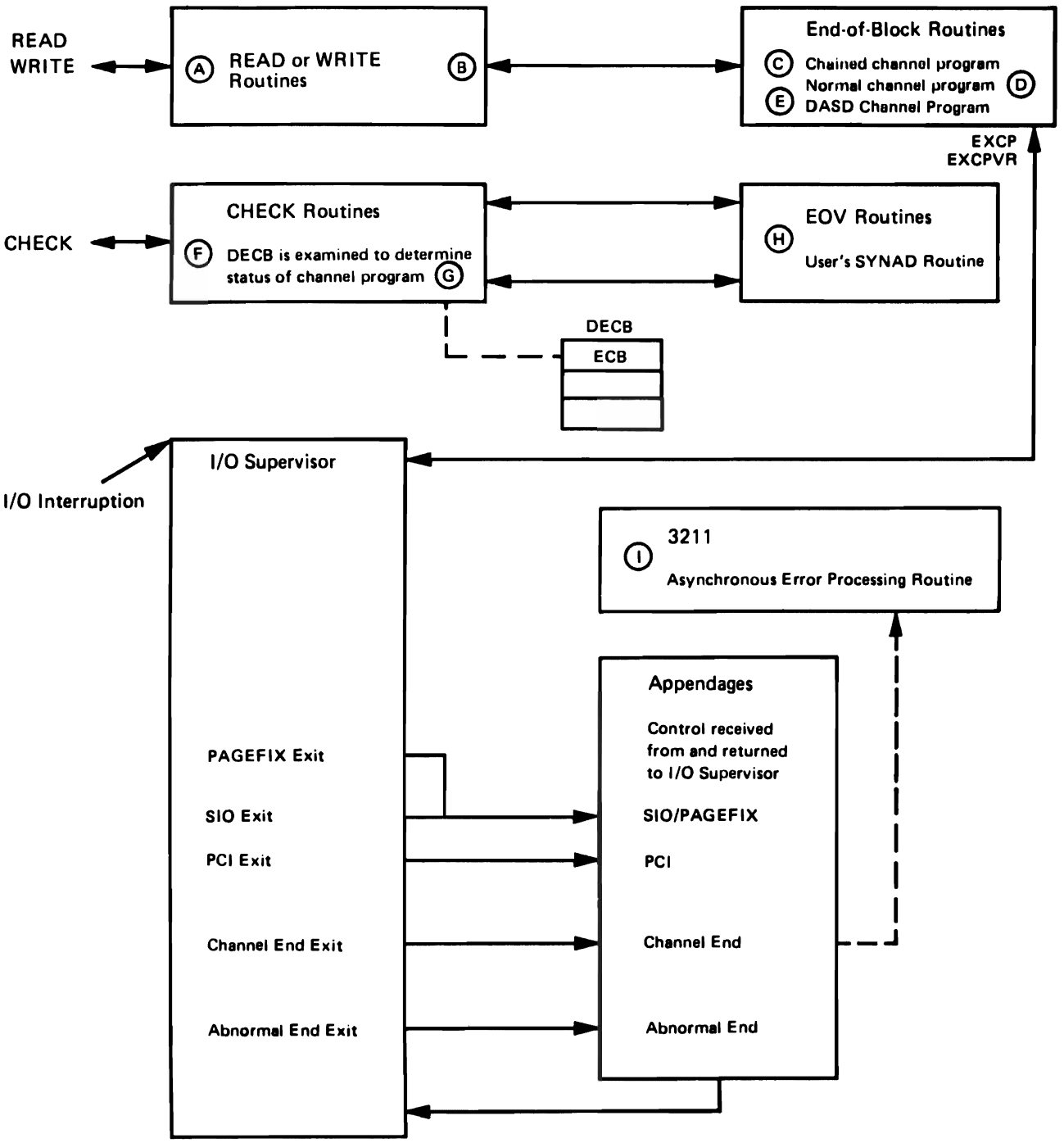
DIAGRAM F: QSAM FLOW OF CONTROL



Notes for Diagram F

(A)	A synchronizing-and-error-processing routine receives control when another full input buffer is needed or if a new empty output buffer is needed.
(B)	An end-of-block routine receives control when an input buffer is empty or an output buffer is full.
(C)	The end-of-block routine attempts to add the present channel program to the last one in the chain of scheduled channel programs. If successful, control returns to the processing program. If unsuccessful, control is passed to the I/O supervisor by an EXCP instruction.
(D)	For normal channel-program scheduling, the routine passes control to the I/O supervisor by an EXCP instruction to cause scheduling of the buffer.
(E)	Direct-access processing end-of-block modules attempt to add another IOB to the IOB chain. If that is successful, control returns to the processing program. If that is not successful, control is passed to the EXCP interface, then to the I/O supervisor, by issuing an EXCPVR instruction.
(F)	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 EOV routine. Control is passed to the EOV Routine by using an SVC 55 instruction in the event that an end-of-volume or a permanent error condition is detected. Refer to Figure 35, "QSAM Control Blocks," for a diagram of the relationship of the IOBs to the other QSAM control blocks.
(G)	The flow of control is described in Diagram H.
(H)	This routine receives control by being scheduled for execution by abnormal-end appendage IGG019C3, IGG019CU, IGG019FR or IGG019V6. Control is passed to the processing program through the supervisor.

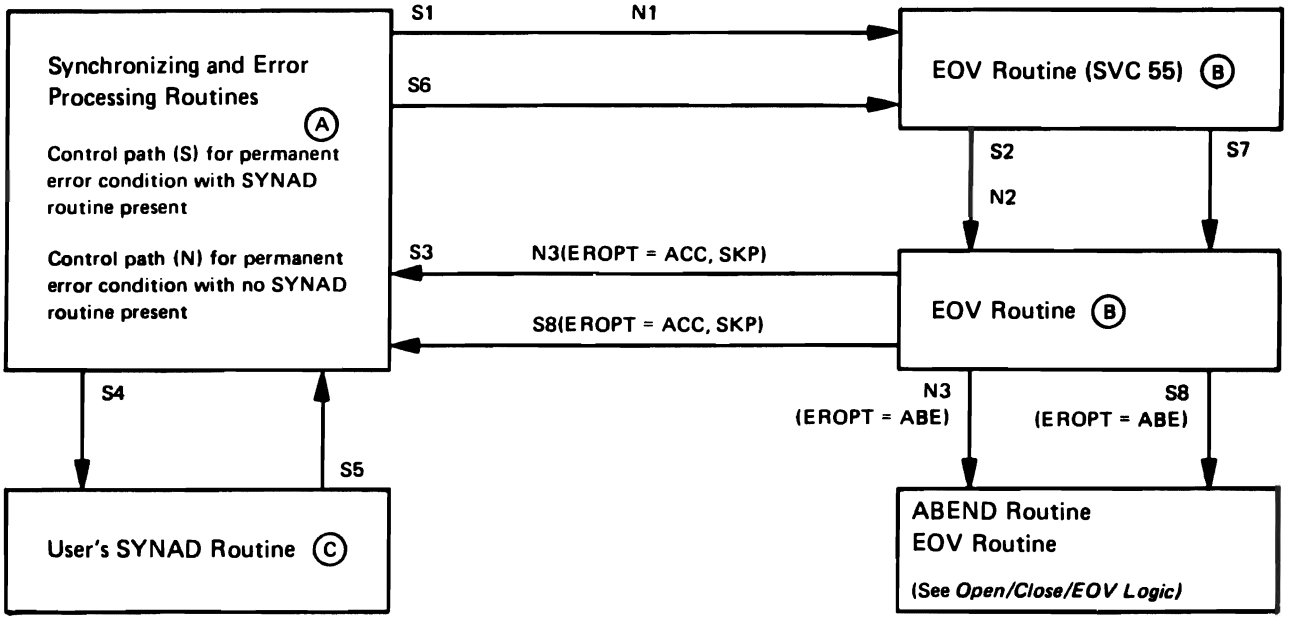
DIAGRAM G: BSAM/BPAM FLOW OF CONTROL

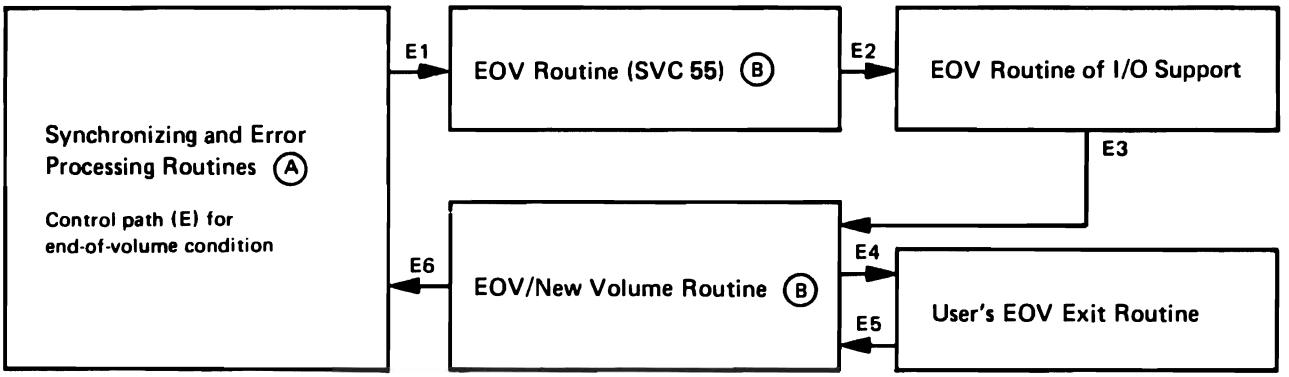


Notes for Diagram G

(A)	A READ or a WRITE routine receives control after a READ or WRITE macro instruction is issued by a processing program.
(B)	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.
(C)	The end-of-block routine attempts to add the present channel program to the last one in the chain of scheduled channel programs. If successful, control returns to the processing program. If unsuccessful, control is passed to the I/O supervisor by an EXCP instruction. These routines are described in the QSAM portion of the manual. See Figure 7.
(D)	For normal channel program scheduling, the routine passes control to the I/O supervisor by an EXCP instruction to cause scheduling of the buffer. The end-of-block routines are described in the QSAM portion of the manual. See Figure 5.
(E)	Direct-access processing end-of-block modules attempt to add another IOB to the IOB chain. If that is successful, control returns to the processing program. If that is not successful, control is passed to the EXCP interface, then to the I/O supervisor, by issuing an EXCPVR instruction.
(F)	A CHECK routine receives control from the processing program via a CHECK macro instruction.
(G)	A CHECK routine returns control to the processing program if the channel program executes normally (without errors). See Figure 36 BSAM Control Blocks for a diagram of the relationship of the DECB to the other BSAM control blocks.
(H)	The flow of control is described in Diagram I.
(I)	This routine receives control by being scheduled for execution by abnormal-end appendage IGG019C3, IGG019CU, IGG019FR or IGG019V6. Control is passed to the processing program through the supervisor.

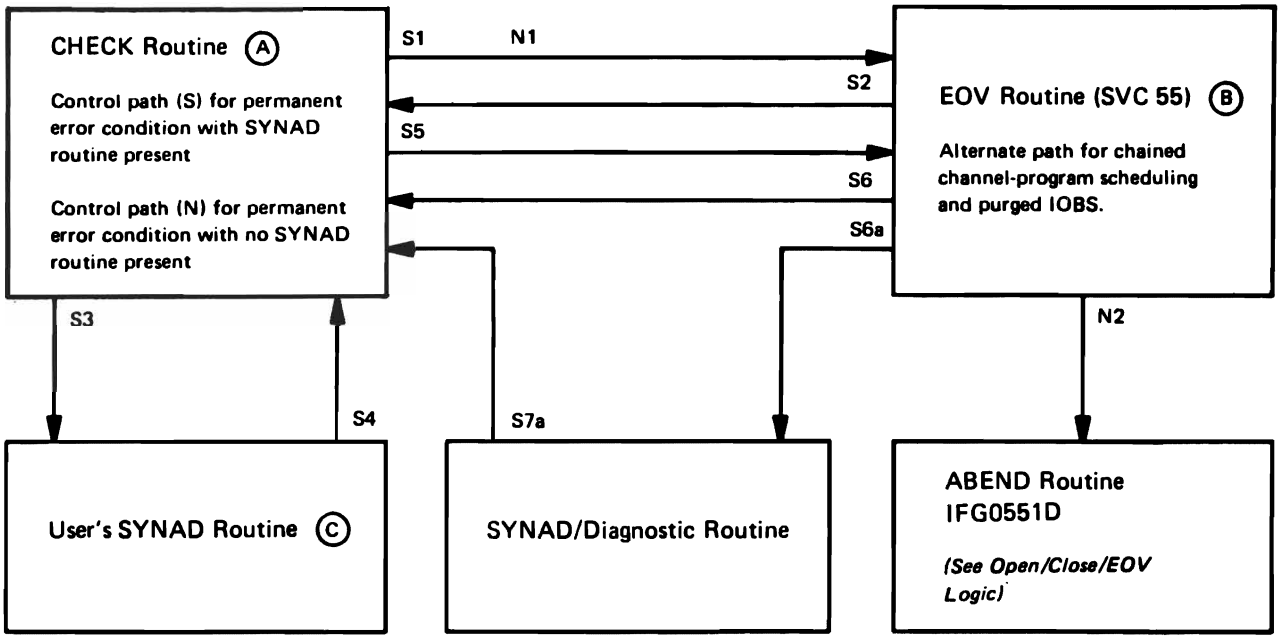
DIAGRAM H: QSAM FLOW OF CONTROL WITH EOV ROUTINES

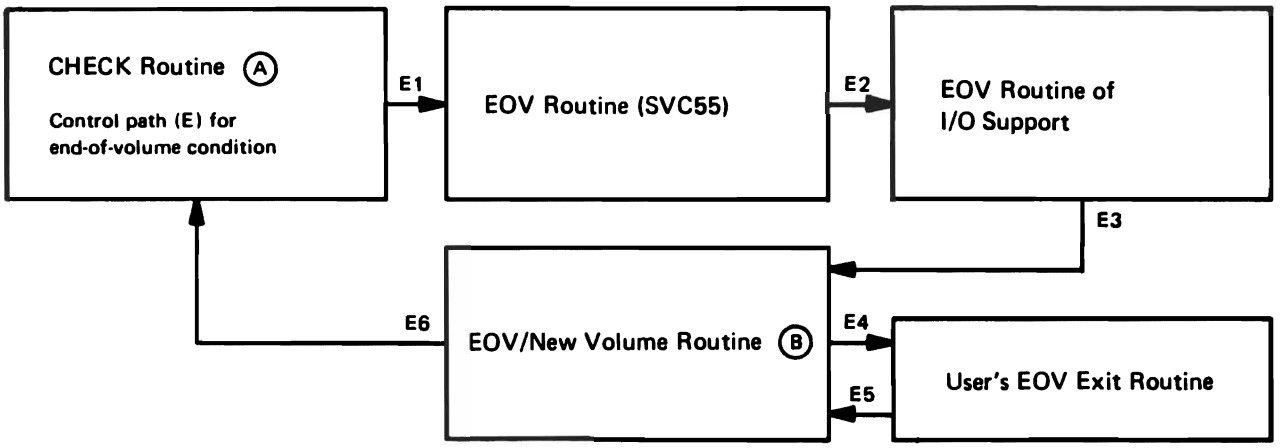




A	Descriptive information on these routines is located in "Synchronizing and Error Processing Routines." See Figures 11 and 12.
B	See ***OPEN/CLOSE/EOV Logic***.
C	The user's SYNAD routine is described in **Data Administration Guide**.

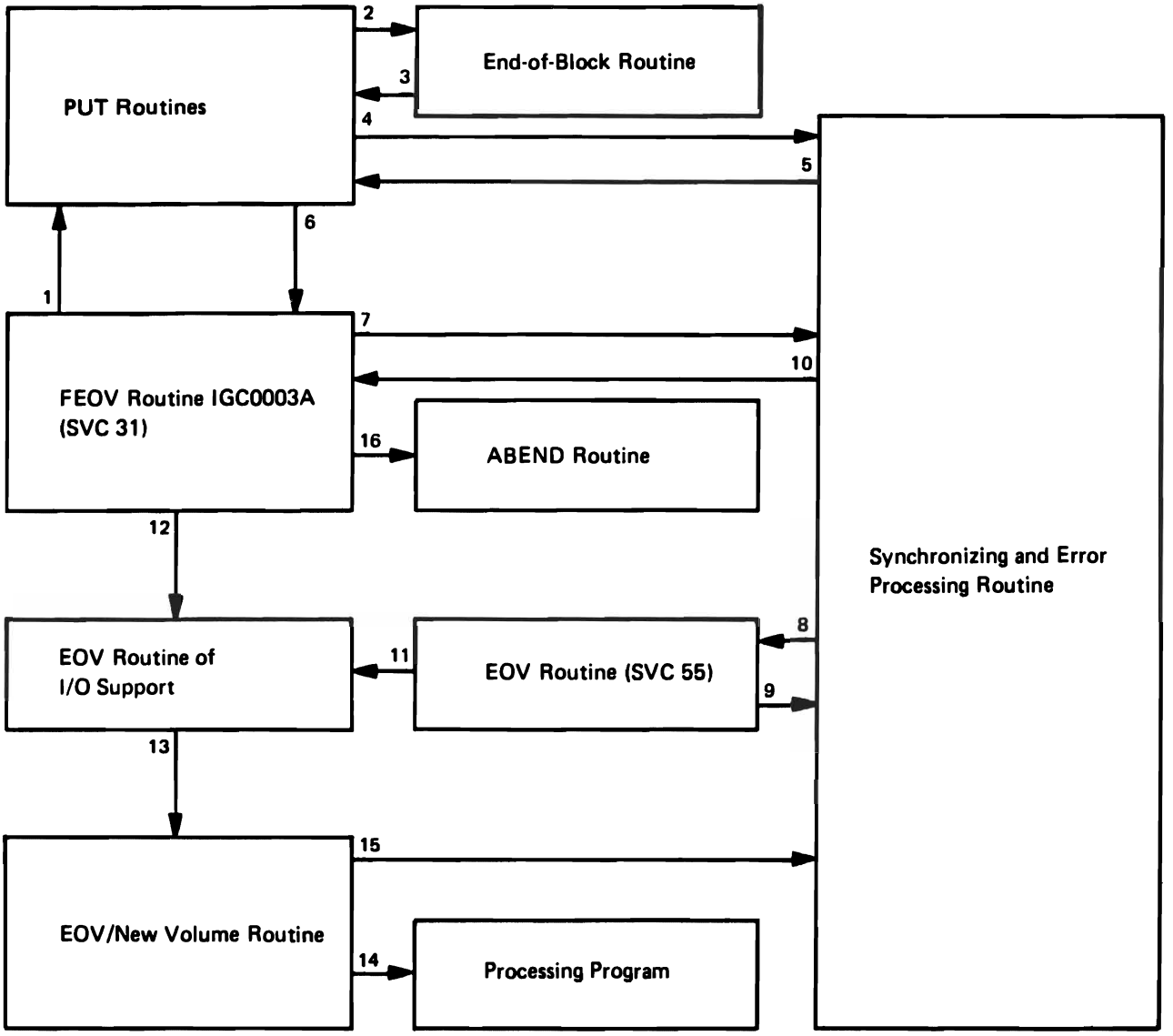
DIAGRAM I: BSAM FLOW OF CONTROL WITH EOV ROUTINES





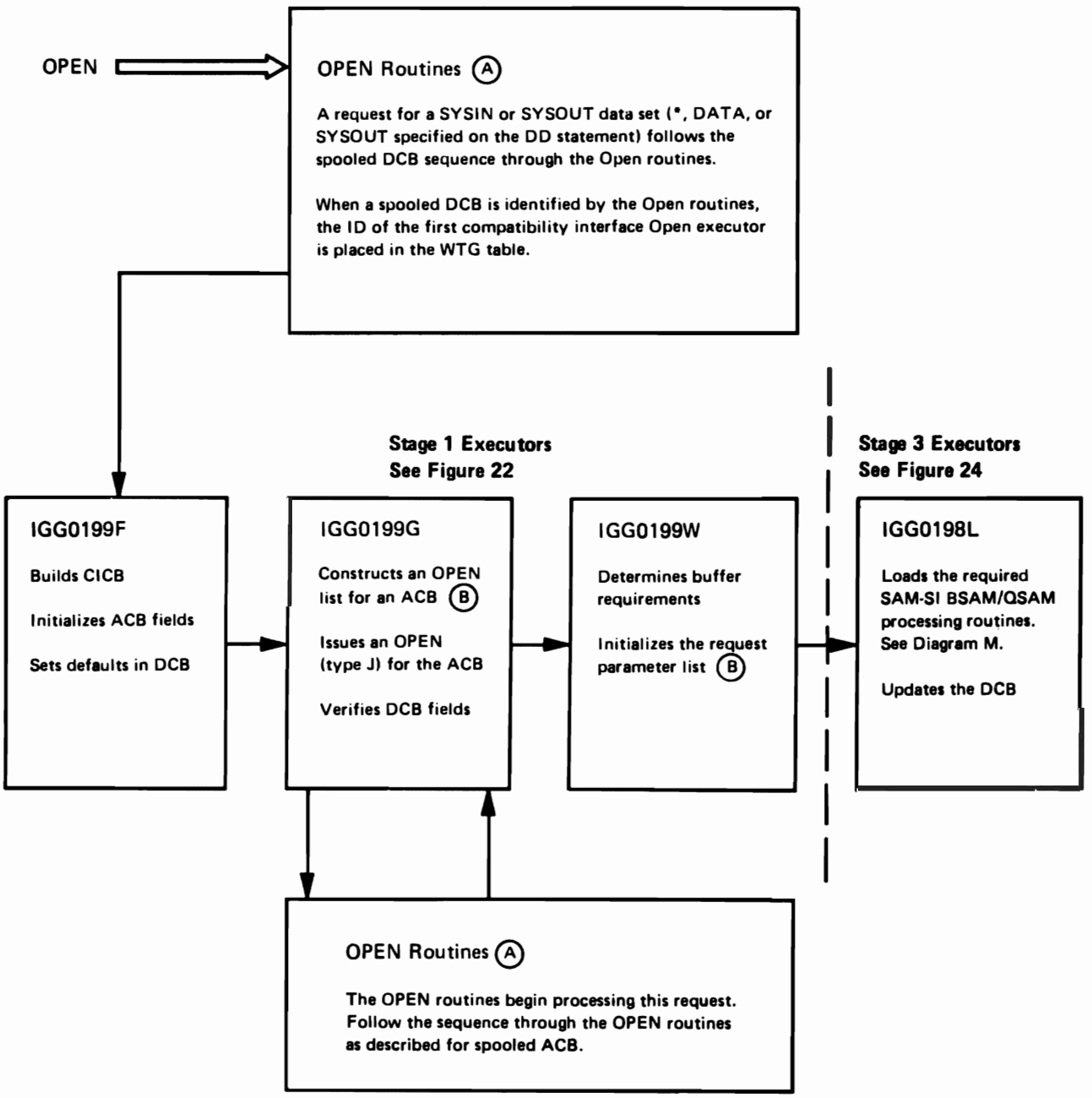
(A)	Descriptive information on the Check routines is located in the "Method of Operation" section under "Basic Sequential Access Method Routines." See Figure 17.
(B)	See ***OPEN/CLOSE/EOV Logic***.
(C)	The user's SYNAD routine is described in **Data Administration Guide**.

DIAGRAM J: QSAM OPERATION WITH FEOV ROUTINE



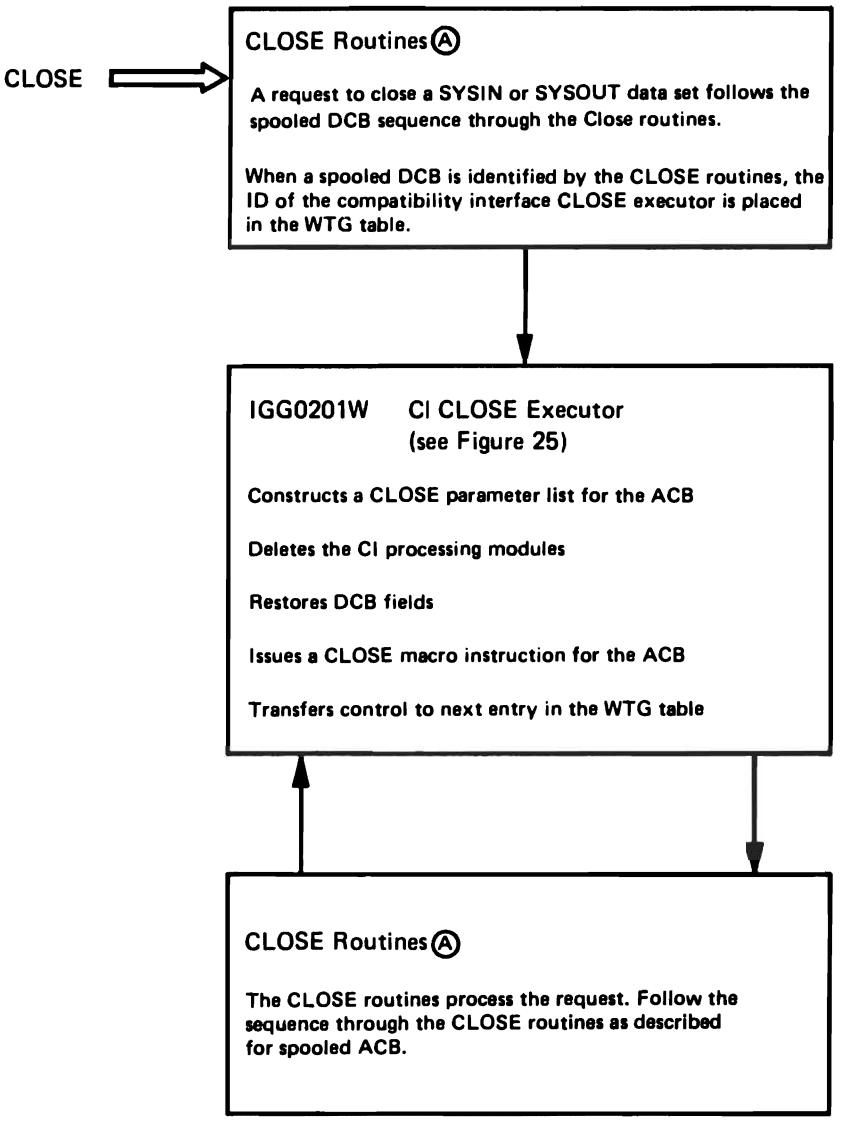
Condition	Sequence of Control
1	1,2,3,6,12,13,14
2	1,2,3,6,7,8,9,10,16
3	1,2,3,6,7,8,11,13,15,10,12,13,14
4	1,2,3,4,5,6,12,13,14
5	1,2,3,4,8,9,10,16
6	1,2,3,4,5,6,7,8,9,10,16
7	1,2,3,4,8,11,13,15,10,12,13,14
8	1,2,3,4,5,6,7,8,11,13,15,10,12,13,14

DIAGRAM K: OPEN PROCESSING FOR SAM SUBSYSTEM INTERFACE EXECUTORS



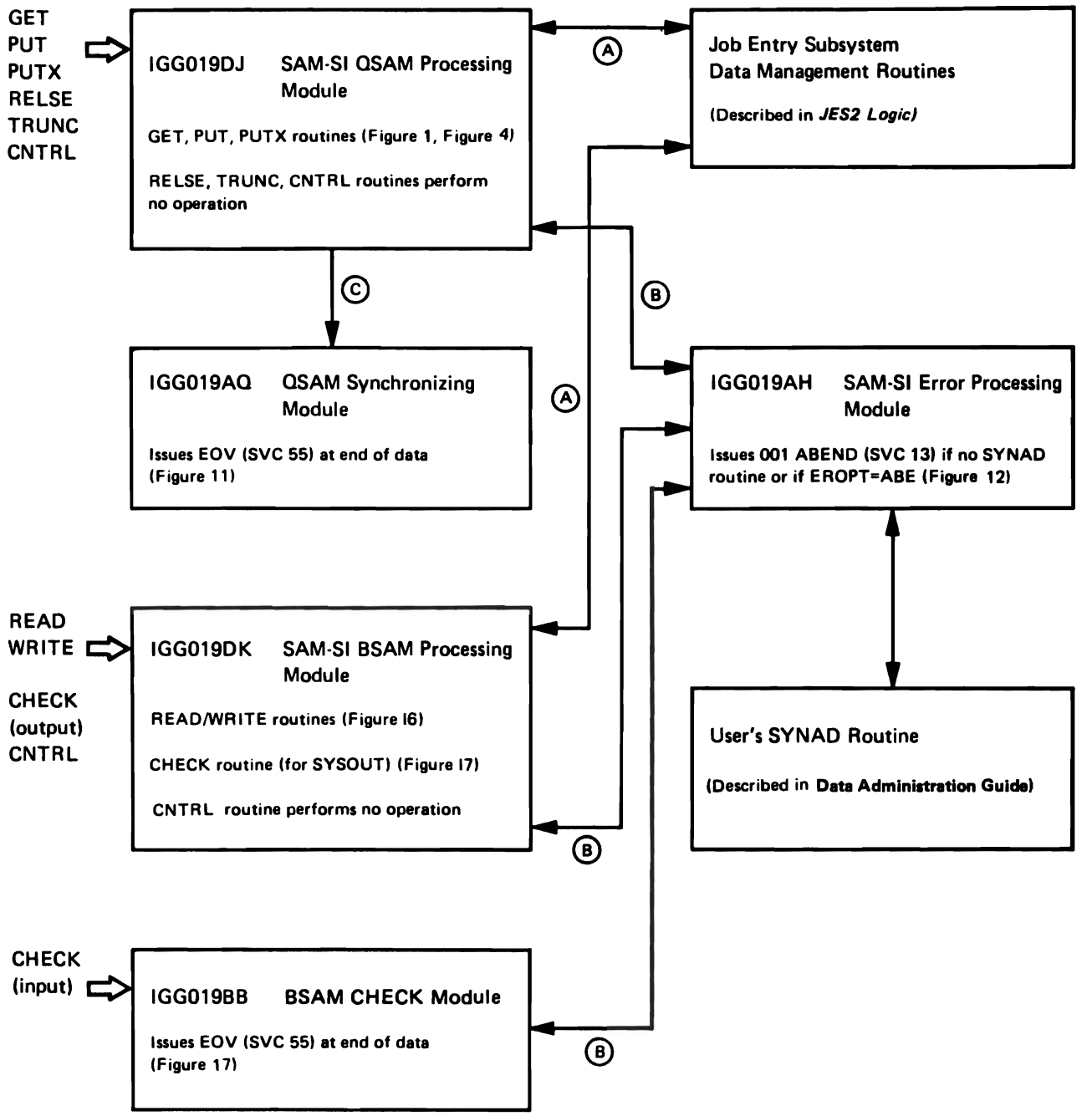
(A)	OPEN routines are described in ***OPEN/CLOSE/EOV Logic***. For information on the WTG and XCTL tables, see the "Access Method Determination" section of the manual.
(B)	The access method control block (ACB) and the request parameter list (RPL) are described in ***Data Areas***.

DIAGRAM L: CLOSE PROCESSING FOR SAM SUBSYSTEM INTERFACE EXECUTORS



Ⓐ	The CLOSE routines are described in ***OPEN/CLOSE/EOV Logic***.

DIAGRAM M: SAM SUBSYSTEM INTERFACE FLOW OF CONTROL FOR SYSIN/SYSOUT DATA SETS



A	Processing program requests translation into GET/PUT for the ACB/RPL that gives control to the Job Entry Subsystem.
B	Permanent error condition returned by the JES
C	End of data condition returned by the JES

DIAGRAM N: FORCE CLOSE PROCESSING

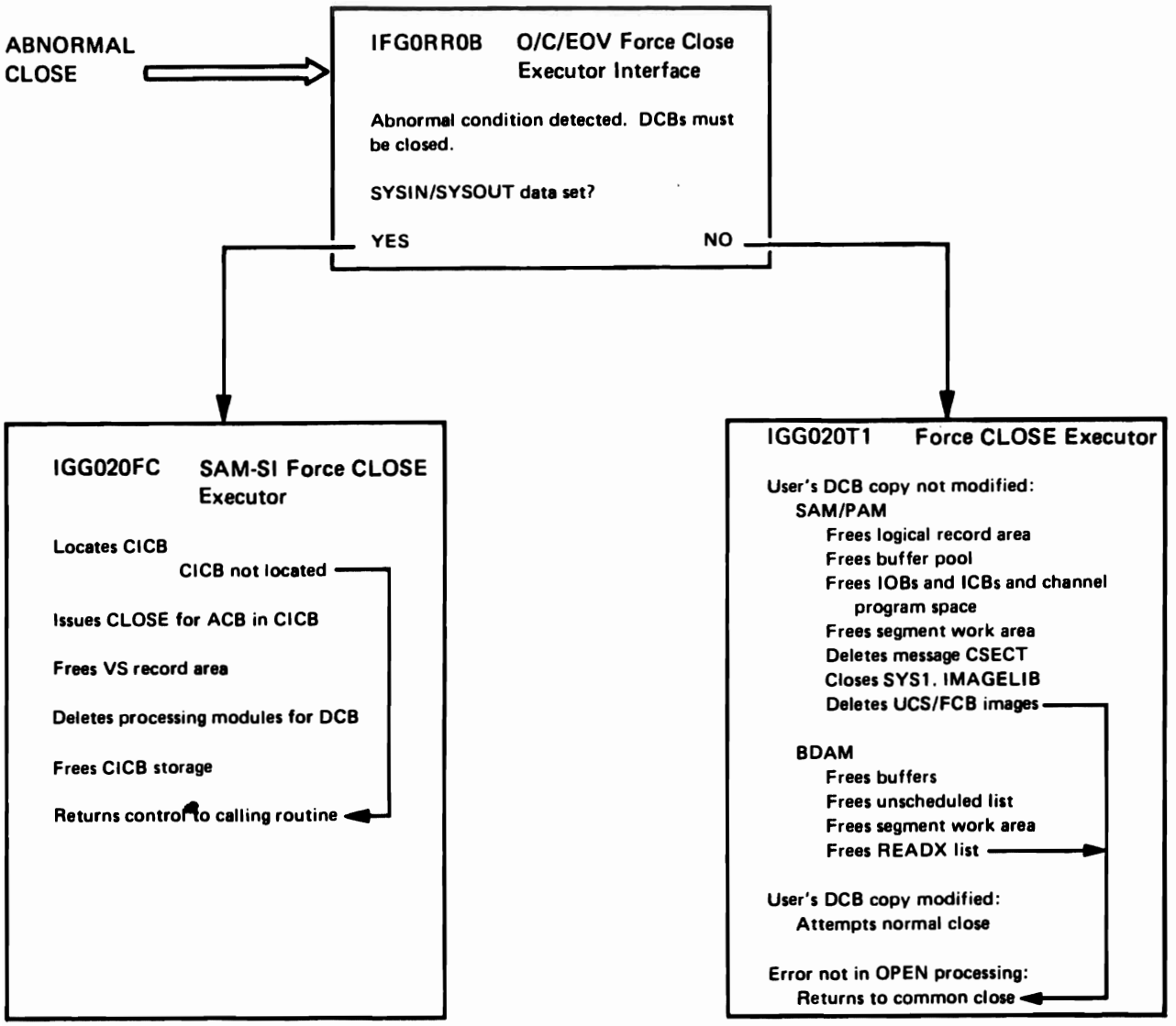
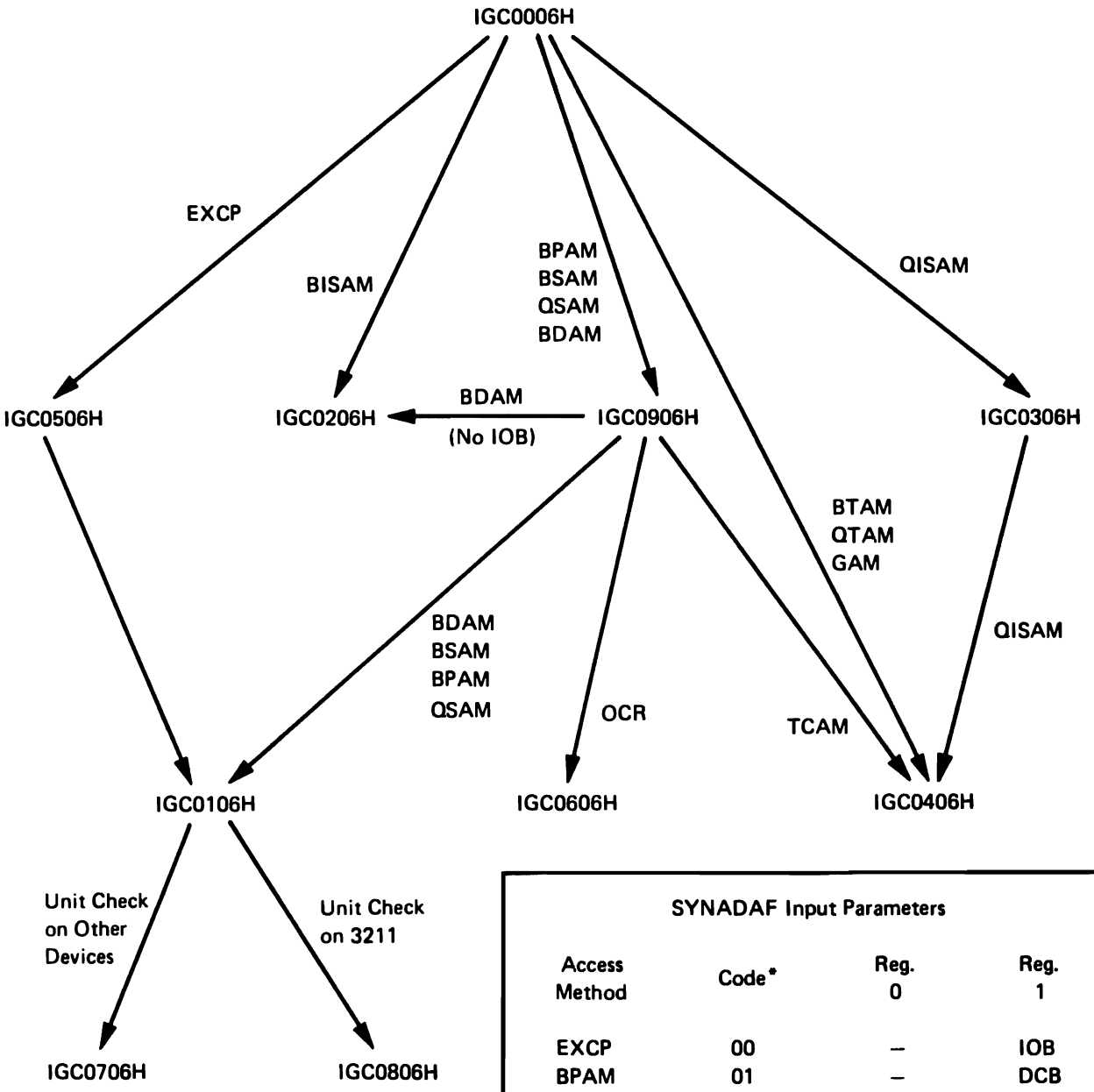


DIAGRAM 0: SYNADAF FLOW OF PROCESSING



All routines can return to user.

SYNADAF Input Parameters

Access Method	Code*	Reg. 0	Reg. 1
EXCP	00	–	IOB
BPAM	01	–	DCB
BSAM	02	–	DCB
QSAM	03	–	DCB
BDAM	04	DECB	DCB
BISAM	05	DECB	DCB
QISAM	06	–	DCB
BTAM	07	–	–
QTAM	08	–	–
GAM	09	–	–
TCAM		–	DCB

*High-order byte, Reg. 15

DIRECTORY

Module Name	Module Type	CSECT Name	SVC Entry	Logic Manual Reference	Module Desc. (Page)
AMDUSRFE	GTRACE format appendage	AMDUSRFE			185
IECBBFB1	Build module	IECBBFB1		Figure 26	149
IECQBFG1	GETPOOL module	IECQBFG1		Figure 26	148
IFG0559C	Problem determination module	IFG0559C			153
IGCT0018	Task recovery routine module	IGCT0018			180
IGCT002D	Task recovery routine module	IGCT002D			181
IGCT002E	Task recovery routine module	IGCT002E			181
IGCT0021	Task recovery routine module	IGCT0021			180
IGCT006H	Task recovery routine module	IGCT006H			182
IGCT0069	Task recovery routine module	IGCT0069			183
IGCT010E	Task recovery routine module	IGCT010E			184
IGCT1081	Task recovery routine module	IGCT1081			183
IGC0002A	STOW module	IGC0002A	21	Figure 20	158
IGC0002D	DEVTYPE module	IGC0002D	24		181
IGC0002E	Control module	IGC0002E	25		181
IGC0006H	SYNADAF module	IGC0006H	68	Diagram 0	182
IGC0006I	Control module	IGC0006I	69		157
IGC0008A	SETPRT routine	IGC0008A	81	Figure 33	168
IGC0010E	IMGLIB routine	IGC0010E	105	Figure 33	184
IGC0010E	IMGLIB executor	ICG0010E	105		184
IGC0106H	SYNADAF module	IGC0106H		Diagram 0	164
IGC0206H	SYNADAF module	IGC0206H		Diagram 0	165
IGC0306H	SYNADAF module	IGC0306H		Diagram 0	165
IGC0406H	SYNADAF module	IGC0406H		Diagram 0	166
IGC0506H	SYNADAF module	IGC0506H		Diagram 0	166
IGC0606H	SYNADAF module	IGC0606H		Diagram 0	166
IGC0706H	SYNADAF module	IGC0706H		Diagram 0	167
IGC0806H	SYNADAF module	IGC0806H		Diagram 0	167
IGC0906H	SYNADAF module	IGC0906H		Diagram 0	167
IGC018	Resident module	IGC018	18	Figure 20	161
		IECPCNVT			161
		IECPRLTV			161
		IECOSCR1			161
IGG019AA	GET module	IGG019AA		Figure 1	7
IGG019AB	GET module	IGG019AB		Figure 1	7
IGG019AC	GET module	IGG019AC		Figure 1	8
IGG019AD	GET module	IGG019AD		Figure 1	9
IGG019AE	GET module	IGG019AE		Figure 3	21
IGG019AF	Synchronizing module	IGG019AF		Figure 11	61
IGG019AG	GET module	IGG019AG		Figure 1	10
IGG019AH	Error-processing module	IGG019AH		Figure 12, Diagram M	66
IGG019AI	PUT module	IGG019AI		Figure 4	28
IGG019AJ	PUT module	IGG019AJ		Figure 4	29
IGG019AK	PUT module	IGG019AK		Figure 4	29
IGG019AL	PUT module	IGG019AL		Figure 4	30
IGG019AM	GET module	IGG019AM		Figure 1,2	10
IGG019AN	GET module	IGG019AN		Figure 1,2	11
IGG019AQ	Synchronizing module	IGG019AQ		Figure 11, Diagram M	63
IGG019AR	Synchronizing module	IGG019AR		Figure 11	64
IGG019AV	DD Dummy	IGG019AV			117
IGG019AX	Save module	IGG019AX		Figure 5,7,9	46
IGG019BA	READ/WRITE module	IGG019BA		Figure 16	99
IGG019BB	CHECK module	IGG019BB		Figure 17, Diagram M	107
IGG019BD	Control module	IGG019BD		Figure 18	111
IGG019BE	Control module	IGG019BE		Figure 18	112
IGG019BH	READ/WRITE module	IGG019BH		Figure 16	99
IGG019BI	CHECK module	IGG019BI		Figure 17	108
IGG019BK	Control module	IGG019BK		Figure 18	113

Module Name	Module Type	CSECT Name	SVC Entry	Logic Manual Reference	Module Desc. (Page)
IGG019BL	Control module	IGG019BL		Figure 18	115
IGG019BN	GET Update module	IGG019BN		Figure 3	22
IGG019BO	GET module	IGG019BO		Figure 1	13
IGG019BP	PUT module	IGG019BP		Figure 6	31
IGG019BQ	Synchronizing module	IGG019BQ		Figure 11	64
IGG019BR	WRITE module	IGG019BR		Figure 16	100
IGG019BS	CHECK module	IGG019BS		Figure 17	109
IGG019BT	Channel end appendage	IGG019BT		Figure 13	82
IGG019BU	READ module	IGG019BU		Figure 16	101
IGG019BX	SIO/pagefix appendage	IGG019BX		Figure 13	76
IGG019BY	SIO/pagefix appendage	IGG019BY		Figure 13	76
IGG019BZ	Channel end and abend appendage	IGG019BZ		Figure 13	88
IGG019B0	BUILDRCD routine	IGG019B0		Figure 26	150
IGG019CA	Control module	IGG019CA		Figure 14,18	95
IGG019CB	Control module	IGG019CB		Figure 14,18	96
IGG019CC	EOB module	IGG019CC		Figure 5	40
IGG019CE	EOB module	IGG019CE		Figure 5	41
IGG019CF	EOB module	IGG019CF		Figure 5	43
IGG019CI	Ch-end and ab-end appendage	IGG019C1		Figure 13	82
IGG019CJ	Ch-end and ab-end appendage	IGG019CJ		Figure 13	83
IGG019CL	SIO appendage	IGG019CL		Figure 13	72
IGG019CT	EOB module	IGG019CT		Figure 5,9	44
IGG019CU	Appendage	IGG019CU		Figure 13	83
IGG019CW	EOB module	IGG019CW		Figure 7	50
IGG019CX	EOB module	IGG019CX		Figure 7	51
IGG019CY	EOB module	IGG019CY		Figure 7	52
IGG019DA	WRITE module	IGG019DA		Figure 16	102
IGG019DB	WRITE module	IGG019DB		Figure 16	102
IGG019DC	CHECK module	IGG019DC		Figure 17	109
IGG019DD	WRITE module	IGG019DD		Figure 16	103
IGG019DJ	GET module	IGG019DJ		Figure 1, Diagram M	33
	PUT module			Figure 4	33
IGG019DK	READ/WRITE module	IGG019DK		Figure 16, Diagram M	109
	CHECK module			Figure 17	109
IGG019EI	Ch-end and ab-end appendage	IGG019EI		Figure 13	85
IGG019EJ	Ch-end and ab-end appendage	IGG019ES		Figure 13	87
IGG019FA	Control module	IGG019FA		Figure 14,18	96
IGG019FB	GET module	IGG019FB		Figure 1	16
IGG019FD	GET module	IGG019FD		Figure 1	17
IGG019FF	GET module	IGG019FF		Figure 1	18
IGG019FG	PUT module	IGG019FG		Figure 4	35
IGG019FJ	PUT module	IGG019FJ		Figure 4	36
IGG019FK	EOB module	IGG019FK		Figure 5	44
IGG019FL	PUT module	IGG019FL		Figure 4	37
IGG019FQ	EOB module	IGG019FQ		Figure 5	44
IGG019FR	Abnormal-end appendage	IGG019FR		Figure 13	94
IGG019FS	Async error module	IGG019FS		Figure 12	67
IGG019FU	EOB module	IGG019FU		Figure 5	46
IGG019JD	Parallel input module	IGG019JD			19
IGG019TC	EOB module	IGG019TC		Figure 5	46
IGG019TV	EOB module	IGG019TV		Figure 9	55
IGG019TW	EOB module	IGG019TW		Figure 7	53
IGG019T2	EOB module	IGG019T2		Figure 9	57
IGG019V6	Appendage	IGG019V6		Figure 13	94
IGG0191A	Stage 1 OPEN executor	IGG0191A		Figure 22, Diagram E	120
IGG0191B	Stage 1 OPEN executor	IGG0191B		Figure 22, Diagram E	121
IGG0191C	Stage 1 OPEN executor	IGG0191C		Figure 22, Diagram E	122
IGG0191G	Stage 2 OPEN executor	IGG0191G		Figure 23, Diagram E	130
IGG0191I	Stage 1 OPEN executor	IGG0191I		Figure 22, Diagram E	122
IGG0191L	Stage 2 OPEN executor	IGG0191L		Figure 23, Diagram E	130
IGG0191M	Stage 2 OPEN executor	IGG0191M		Figure 23, Diagram E	131
IGG0191N	Stage 1 OPEN executor	IGG0191N		Figure 22, Diagram E	123
IGG0191Q	Stage 2 OPEN executor	IGG0191Q		Figure 23, Diagram E	131
IGG0191R	Stage 2 OPEN executor	IGG0191R		Figure 23, Diagram E	132

Module Name	Module Type	CSECT Name	SVC Entry	Logic Manual Reference	Module Desc. (Page)
IGG0191Y	Stage 1 OPEN executor	IGG0191Y		Figure 22, Diagram E	123
IGG01910	Stage 3 OPEN executor	IGG01910		Figure 24, Diagram E	136
IGG01911	Stage 3 OPEN executor	IGG01911		Figure 24, Diagram E	137
IGG01913	Stage 3 OPEN executor	IGG01913		Figure 24, Diagram E	138
IGG01915	Stage 3 OPEN executor	IGG01915		Figure 24, Diagram E	138
IGG01916	Stage 3 OPEN executor	IGG01916		Figure 24, Diagram E	139
IGG0193B	Stage 2 & 3 OPEN executor			Figure 23, Diagram E	140
IGG0193I	Stage 1 OPEN executor	IGG0193I		Figure 22, Diagram E	124
IGG0196A	Stage 1 OPEN executor	IGG0196A		Figure 22, Diagram E	124
IGG0196B	Stage 1 OPEN executor	IGG0196B		Figure 22, Diagram E	125
IGG0196I	Stage 2 OPEN executor	IGG0196I		Figure 22, Diagram E	126
IGG0196K	Stage 2 OPEN executor	IGG0196K		Figure 23	132
IGG0196Q	Stage 1 OPEN executor	IGG0196Q			126
IGG0196S	Stage 2 TS module[1]	IGG0196S			
IGG0197L	Stage 1 OPEN executor	IGG0197L		Figure 22, Diagram E	126
IGG0197M	Stage 2 OPEN executor	IGG0197M		Figure 22, Diagram E	127
IGG0197N	Stage 2 OPEN executor	IGG0197N		Figure 23, Diagram E	133
IGG0197P	Stage 2 OPEN executor	IGG0197P		Figure 23, Diagram E	133
IGG0197Q	Stage 2 OPEN executor	IGG0197Q		Figure 23, Diagram E	134
IGG0197V	Stage 2 OPEN executor	IGG0197V		Figure 23	134
IGG0198L	Stage 3 OPEN executor	IGG0198L		Figure 24, Diagram K	140
IGG0199F	Stage 3 OPEN executor	IGG0199F		Figure 22, Diags E,K	127
IGG0199G	Stage 3 OPEN executor	IGG0199G		Figure 22, Diags E,K	127
IGG0199L	Stage 2 OPEN executor	IGG0199L		Figure 23, Diagram E	134
IGG0199W	Stage 1 OPEN executor	IGG0199W		Figure 22, Diags E,K	128
IGG020FC	SAM-SI Force Close executor	IGG020FC		Diagram N	146
IGG020T1	SAM/PAM/DAM force CLOSE executor	IGG020T1		Diagram N	146
IGG0201A	CLOSE executor	IGG0201A		Figure 25	141
IGG0201B	CLOSE executor	IGG0201B		Figure 25	142
IGG0201P	CLOSE executor	IGG0201P		Figure 25	143
IGG0201R	CLOSE executor	IGG0201R		Figure 25	143
IGG0201W	CLOSE executor	IGG0201W		Figure 25	144
IGG0201X	CLOSE executor	IGG0201X		Figure 25	144
IGG0201Y	CLOSE executor	IGG0201Y		Figure 25	145
IGG0201Z	CLOSE executor	IGG0201Z		Figure 25	145
IGG021AB	STOW module	IGG021AB		Figure 20	160
IGG0210A	STOW module	IGG0210A		Figure 20	159
IGG08101	SETPRT routine	IGG08101		Figure 33	171
IGG08102	SETPRT routine	IGG08102		Figure 33	172
IGG08103	SETPRT routine	IGG08103		Figure 33	&8103
IGG08104	SETPRT routine	IGG08104		Figure 33	173
IGG08105	SETPRT routine	IGG08105		Figure 33	174
IGG08108	SETDEV routine	IGG08108		Figure 33	175
IGG08110	SETPRT executor	IGG08110		Figure 33	175
IGG08111	SETPRT executor	IGG08111		Figure 33	176
IGG08112	SETPRT executor	IGG08112		Figure 33	176
IGG08113	SETPRT executor	IGG08113		Figure 33	177
IGG08114	SETPRT executor	IGG08114		Figure 33	177
IGG08115	SETPRT executor	IGG08115		Figure 33	177
IGG08116	SETPRT executor	IGG08116		Figure 33	177
IGG08117	SETPRT executor	IGG08117		Figure 33	178
IGXMSG01	MSGDISP message text	IGXMSG01			
IGX00030	MSGDISP module	IGX00030	109		115
IGX00031	SYNCDEV module	IGX00031	109		116
IGX00032	NOTE/POINT module	IGX00032	109		116

[1] See ACF/TCAM Diagnosis Guide and ACF/TCAM Diagnosis Reference.

The modules of OPEN, CLOSE, STOW and SYNADAF are link-edited into SYS1.LPALIB during system generation, by macro SCIEC4DI, according to the following list. Each CSECT is a separate module in the distribution library and in microfiche. For IGG0191A, IGG01911, and IGG0201Z, each CSECT name is also an alias for the load module.

Load module	CSECTs
IGG0191A	IGG0191A, IGG0191B, IGG0191I, IGG0191N, IGG0191Y, IGG0193I, IGG0196A, IGG0196B, IGG0196I,
IGG0201Z	IGG0201A, IGG0201B, IGG0201X, IGG0201Y, IGG0201Z
IGC0002A	IGC0002A, IGG021AB, IGG0210A
IGC0006H	IGC0006H, IGC0106H, IGC0206H, IGC0306H, IGC0406H, IGC0506H, IGC0606H, IGC0706H, IGC0806H, IGC0906H
IGG01911	IGG0191C, IGG0191G, IGG0191Q, IGG0191R, IGG0191O, IGG01911, IGG01913, IGG01915, IGG01916, IGG0196K
IGG0193B	IGG019BK, IGG019BX, IGG019BY, IGG019BZ, IGG019TV, IGG019T2, IGG0193B

IOB EXTENSION (USED WITH SAM EXCPVR)—IGGIOBEX

Offset	Length	Name	Description
IOBSEEK+8			
40(X'28')	24	IOBCCSV	Channel-program area
40(X'28')	8	IOBCCW1	First channel command word (CCW):
40(X'28')	1	IOBCCW1O	Command code
41(X'29')	3	IOBCCW1A	Address
44(X'2C')	1	IOBCCW1F	Flags
45(X'2D')	1		Unused
46(X'2E')	2	IOBCCW1L	Data length
48(X'30')	8	IOBCCW2	Second channel command word (CCW):
48(X'30')	1	IOBCCW2O	Command code
49(X'31')	3	IOBCCW2A	Address
52(X'34')	1	IOBCCW2F	Flags
53(X'35')	1		Unused
54(X'36')	2	IOBCCW2L	Data length
48(X'30')	8	IOBCNT	Count field, in the form CCHHRKDD
48(X'30')	5	IOBCNTCH	CCHHR: a 5-byte argument used by SEARCH CCWs
48(X'30')	2	IOBCNTCC	CC: cylinder value
50(X'32')	2	IOBCNTHH	HH: track value
52(X'34')	1	IOBCNTR	R: record number
53(X'35')	3	IOBCNTKD	KDD part of the count field
53(X'35')	1	IOBCNTK	K: length of the record's key area
54(X'36')	2	IOBCNTDD	DD: length of the record's data area
56(X'38')	0	IOBCCWND	End of the CCW chain
56(X'38')	8	IOBTRKOV	Track overflow area
56(X'38')	2	IOBLFST	Length of the first segment
56(X'38')	1	IOBSECTO	Sector value
57(X'39')	1		Reserved
58(X'3A')	1	IOBNINCL	Number of segments in the first segment's cylinder
59(X'3B')	1	IOBNMID	Number of track capacity middle segments
60(X'3C')	2	IOBLMID	Data length of track capacity middle segments
62(X'3E')	2	IOBLLST	Data length of the last segment—this is zero if there is only one segment.

SEQUENTIAL ACCESS METHOD BLOCK—IGGSAMB

The SAMB is used to control the accessing of data sets that reside on direct access storage devices.

Offset	Length	Name	Description
0(X'0')	0	SAMBXXX	SAMB prefix
	...1 111.	SAMAXBUF	30 maximum number of buffers that can be scheduled per SIO
	.111 1.1.	SAMAXIDA	122 maximum number of IDAWS in SAMB
	..1.	SAMINIDA	32 minimum number of IDAWS in SAMB
	.... 11.1	SAMINBUF	13 minimum number of buffers that may determine SAMB size
0(X'0')	8	SAMPRFIX	Prefix for SAMB
0(X'0')	1	SAMPSUB	Subpool of SAMB
1(X'1')	3	SAMPLENG	Length of the SAMB + SAMB prefix
4(X'4')	4		Unused
8(X'8')	0	SAMB	
8(X'8')	72	SAMIOB	IOB for use by EXCPVR
8(X'8')	8		IOB PREFIX
8(X'8')	4		
12(X'C')	4		EBCDIC 'SAMB'
16(X'10')	64		IOB
80(X'50')	4	SAMICQL	Length of ICQE and IOBS
84(X'54')	4	SAMRQEA	PTR to RQE
88(X'58')	4	SAMCNTA	PTR to next available count area
92(X'5C')	4	SAMCCWA	PTR to next available real CCW
96(X'60')	4	SAMIDAWA	PTR to next available IDAW
100(X'64')	4	SAMPGFXA	PTR to next available PGFIX entry
104(X'68')	4	SAMIOBP	Previously posted IOB
108(X'6C')	4	SAMPSTWD	Post code word
108(X'6C')	1	SAMPSTCD	Post code
109(X'6D')	3		Remainder of post code word
112(X'70')	1	SAMSMFCT	SMF count
113(X'71')	1	SAMSEGCT	Segment count
114(X'72')	0	SAMCCWC	CCW count for each segment
114(X'72')	61	SAMCCWCT	Same as SAMCCWC
175(X'AF')	1	SAMOPNID	Open type indicator
	1	SAMOPNIN	1 open for input
	.... ..1.	SAMOPNOT	2 open for output
	11	SAMOPNUP	3 open for update
	.... .1..	SAMOPNTO	4 open for TRK overflow output
	.... .1.1	SAMOPNOI	5 open for OUTIN or INOUT
176(X'B0')	1	SAMKEY	User key
177(X'B1')	1	SAMFLAG1	Flag byte one
	1...	SAMFOUND	X'80' something found
	.1..	SAMERROR	X'40' dynamic error bit
	..1.	SAMFIX	X'20' a page fix is needed
	...1	SAMFREE	X'10' a page free is needed
	.... 1...	SAMRST	X'08' the CHAN PROG has been broken
	.... .1..	SAMPUTX	X'04' this is a PUTX request
	.... ..1.	SAMBSWR	X'02' this is a BSAM update WRT REQ
	1	SAMERIGN	X'01' ignore SAMERROR
178(X'B2')	1	SAMFLAG2	
	1...	SAMACT	X'80' a request is active
	.1..	SAMSIO	X'40' SIO is to ignore this request
	..1.	SAMVRDEV	X'20' virtual device
	...1	SAMSKSVD	X'10' IOBSEEK saved in SAMSKSAV for VBS LRI update
	.... 1...	SAMEOE	X'08' EOE processing has occurred
	.... .1..	SAMSEGMT	X'04' Segmenting of a VBS LRI record is occurring

Offset	Length	Name	Description
	.... ..1.	SAMRPS	X'02' RPS device
	1	SAMCLRTM	X'01' PAGEFIX appendage issued CALLRTM-SIO take ignore exit
179(X'B3')	1	SAMFLAG3	
	1...	SAMPSTNX	X'80' next IOB POST41-UNIT exception
	.1..	SAMSCHPR	X'40' search previous
	..1.	SAMRDSK	X'20' read skip needed
	...1	SAMSMF	X'10' call SMF for EXCP count
	.... 1...	SAMLRF	X'08' device supports ECKD
180(X'B4')	1	SAMFLGSV	Save area for SAMFLAG3 when SAMSEGMT is on
181(X'B5')	1	SAMOFLGS	Flag byte two set by open
	1...	SAMUPD	X'80' Update open
	.1..	SAMWCHK	X'40' WRT validity check
	..1.	SAMTRKO	X'20' track overflow
	...1	SAMVEQR	X'10' V=R region
	.... 1...	SAMONE	X'08' only one buffer acquired
	.... .1..	SAMBFTKR	X'04' BFTEK=R option requested
182(X'B6')	1	SAMFRRFG	SAM FRR flags
	1...	SAMCV	X'80' SAMPGFXS is valid
	.1..	SAMIP	X'40' FRR must process PAGEFIX list
	..1.	SAMFFIP	X'20' free/fix in process
	...1	SAMFRIP	X'10' free in process
	.... 1...	SAMFIIP	X'08' fix in process
	.... .1..	SAMTOUCH	X'04' validity checking buffer
	.... ..1.	SAMCPCK	X'02' buffer address INVALID-CH.PGM check will be mapped as channel protection check
183(X'B7')	1	SAMSPECT	Write special count
184(X'B8')	1	SAMRTNOF	Return offset from appendages
185(X'B9')	3	SAMRESV1	Reserved
188(X'BC')	4	SAMPFECB	ECB for page fix request
188(X'BC')	1	SAMPFECC	ECB condition code
189(X'BD')	3	SAMPECBB	
192(X'C0')	4	SAMODEBA	Old DEB address for QSAM VBS LRI multivolume update
196(X'C4')	4	SAMFRRPT	Address of a 6-word parameter area returned from SETFRR
200(X'C8')	4	SAMREAL	Offset to get real address of SAMB
204(X'CC')	11	SAMSEEKS	Dual use seek area
204(X'CC')	8	SAMSEEK	SAMSEEK MBBCCHHR
204(X'CC')	1	SAMSEEKM	SAMSEEK M
205(X'CD')	2	SAMSEKBB	SAMSEEK BB
207(X'CF')	5	SAMCNT	SAMCNT CCHHR
207(X'CF')	5	SAMCCHHR	CCHHR for both
207(X'CF')	4	SAMCCHH	CCHH for both
211(X'D3')	1	SAMR	R field for both
212(X'D4')	3	SAMCTKDD	SAMCNT KDD
212(X'D4')	1	SAMCTKEY	SAMCNT key
213(X'D5')	2	SAMCTDD	SAMCNT DD
215(X'D7')	1	SAMSECT	Sector value for RPS
216(X'D8')	8	SAMSKSAV	Saved IOBSEEK if SAMSKSVD=ON
224(X'E0')	8		
224(X'E0')	80	SAMPGFX	Space for 10 PFGIX list entries
304(X'130')	80	SAMPGFXS	Space for 10 PGFIX list entries
384(X'180')	120	SAMSAV	Work area (compiler)
504(X'1F8')	4	SAMLRPTR	Address of current locate work area
508(X'1FC')	48	SAMLRPRM	Space for 3 locate record parameter lists
556(X'22C')	4	SAMCCPTR	Address of SAMCCW (follows and contiguous to SAMTIC2)

Offset	Length	Name	Description
560(X'230')4		SAMCTPTR	Address of SAMCNTS (follows and contiguous to SAMCCW)
564(X'234')4		SAMIDPTR	Address of SAMIDAW (follows and contiguous to SAMCNTS)
568(X'238')4		SAMIEPTR	Address of SAMIDAND (follows and contiguous to SAMIDAW)
572(X'23C')4			Reserved
576(X'240')8		SAMPROLG	CHAN PROG prolog
576(X'240')8	258	SAMSS	Set sector CCW
576(X'240')1		SAMSSO	Command code
577(X'241')3		SAMSSA	ADDR
580(X'244')1		SAMSSF	Flags
581(X'245')1			
582(X'246')2		SAMSSC	Count
582(X'246')1		SAMSECT2	Special sector value
583(X'247')1			
584(X'248')8	260	SAMSID	Search ID CCW
584(X'248')1		SAMSIDO	Command code
585(X'249')3		SAMSIDA	IOBSEEK ADDR
588(X'24C')1		SAMSIDF	Flags
589(X'24D')3			
592(X'250')8	268	SAMLRCCW	Locate record CCW
592(X'250')8		SAMTIC1	TIC CCW
592(X'250')1		SAMTIC1O	Command code
593(X'251')3		SAMTIC1A	TIC address
596(X'254')4			Filler
600(X'258')8		SAMTIC2	TIC CCW
600(X'258')1		SAMTIC2O	Command code
601(X'259')3		SAMTIC2A	TIC address
604(X'25C')4			Filler

SAMCCW is pointed to by SAMCCPTR.

SAMCCW follows and is contiguous to SAMTIC2.

Offset	Length	Name	Description
608(X'260')	LL*8	SAMCCW	Space for CCWS for up to SAMAXBUF buffers

SAMCNTS is pointed to by SAMCTPTR.

SAMCNTS follows and is contiguous to SAMCCW.

Offset	Length	Name	Description
0(X'0')	MM*8	SAMCNTS	Space for up to SAMAXBUF+1 count fields
0(X'0')	0	SAMCCWND	End of SAMCCW
0(X'0')	4	SAMRFLST	Start of refill channel program for update
4(X'4')	4	SAMRFLRD	ADDR of read data or read CT, read data, or read data skip, read count, read data
8(X'8')	MM-1 *8		Space for up to SAMAXBUF count fields

SAMIDAW is pointed to by SAMIDPTR.

SAMIDAW follows and is contiguous to SAMCNTS.

Offset	Length	Name	Description
0(X'0')	0	SAMIDAW	Space for SAMINIDA to SAMAXIDA IDAWS
0(X'0')	NN*4		IDAWS

SAMIDAND is pointed to by SAMIEPTR.

SAMIDAND follows and is contiguous to SAMIDAW.

Offset	Length	Name	Description
0(X'0')	0	SAMIDAND	End of SAMIDAW

INTERRUPT CONTROL QUEUE ELEMENT—IGGICQE

Offset	Length	Name	Description
0(X'0')	4	ICQECB	The ECB pointed to by the IOB contained in the SAMB
4(X'4')	4	ICQIOBAD	Address of the IOB in the SAMB
8(X'8')	4	ICQFIRST	Address of the first user IOB
12(X'C')	4	ICQENDA	Address of the last user IOB
12(X'C')	1	ICQFLG	Flag byte:
	1...	ICQEXND	EXCPVR processing is needed
	.xxx xxxx		Reserved
13(X'D')	3	ICQENDAD	Address of the last user IOB
16(X'10')	4	ICQFSTQ	Address of first queued IOB
20(X'14')	1	ICQMAXQ	Maximum number of IOBs on queues
21(X'15')	1	ICQNOQ	Current number on queue
22(X'16')	1	ICQSAVQ	Area for EOV to save ICQMAXQ
23(X'17')	1		Reserved
24(X'18')	8	ICQSAVCT	Save area for the MBBCCHHR used in module IGG019T2:
24(X'18')	1	ICQSVCTM	M value
25(X'19')	2	ICQSVCTB	BB value
27(X'1B')	2	ICQSVCTC	CC value
29(X'1D')	2	ICQSVCTH	HH value
31(X'1F')	1	ICQSVCTR	R value
32(X'20')	72	ICQSAV	Save area for end-of-block processing (EOB)

MESSAGE CSECT—IGGMSG

The message CSECT contains messages for SAM, PAM, and DAM. It is divided into two parts. The first part is the index; the second part contains the message.

Offset Index	Length	Name	Description
0(X'0')	2	MSGINDLN	Length of index
2(X'2')	2	MSGINDOF	Offset to message entry for first message

Offset Index	Length	Name	Description
4(X'4')	2	MSGIND2	Offset to message entry for second message
n(n)	2	MSGINDn	Offset to message entry for nth message

There are two forms of the message entry, with variables and no-variable.

Message Entry—Variable

Offset Index	Length	Name	Description
0(X'0')	1	MSGOFF	Offset to message from beginning of entry
1(X'1')	1	MSGLNG	Length of message -1 (to allow use in execution of a move)
2(X'2')	1	MSGOFF1	Offset to first variable filled in by the routine that extracts the message and causes it to be printed
3(X'3')	1	MSGOFF2	Offset to second variable
n(n)	1	MSGOFFn	Offset to nth variable
n+1(n+1)	1	MSGTXT	Message text

No-variable messages such as those used in SYNADAF

Offset Index	Length	Name	Description
0(X'0')	1	MSGLNGF	Length of message text
1(X'1')	1	MSGTXTF	Message text

SETPRT WORK AREA (SPW)—IGGSPW

The SETPRT work area is used by the SETPRT executors, and is located (when the SETPRT executors are in control) at the address contained in register 4 (which is in user-key virtual storage).

Offset	Length	Name	Description
0(X'0')	1	SPWOPTSA	Save area for the opcode byte
1(X'1')	1	SPWPARM1	Saved reply flags
2(X'2')	1	SPWFLG1	Reply flags:
	1...	SPWVRFCB	Verify FCB
	xxxx xxxx		Reserved
	1	SPWALIGN	Align forms
3(X'3')	1	SPWFLAG2	Module entry indicator
4(X'4')	1	SPWFLAG3	Flags:
	.1..	SPWLDREQ	UCS/FCB load required
	...1	SPW120RQ	Message IEC120 is required
	x.x. xxxx		Reserved
5(X'5')	1...	SPWFLG4	Flag byte:
	1...	SPWFCBDE	FCB image is loaded and must be deleted

Offset	Length	Name	Description
	.1..	SPWECPAM	EXCP DCB with access method section is present
	..xx xxxx		Reserved
6(X'6')	1	SPWFLG8	Flag byte:
	1...	SPWRETRY	Retry in progress
	.1..	SPWVREND	Last verify line is printing
	..1.	SPWNOMOV	Do not move requested message
	...1	SPWFCBOP	Fill FCBOP as well as FCBID into the UCB
	 xxxx		Reserved
7(X'7')	1	SPWRTRYC	Retry counter

WTOR Prefix, Message Section, and Reply Area—in User Key

Offset	Length	Name	Description
8(X'8')	8	SPWMSGHD	Header section for message area
8(X'8')	4	SPWRPLYA	Data about the reply area
8(X'8')	1	SPWMSGLB	Length of the reply area
9(X'9')	3	SPWRPLYB	Address of the reply area
12(X'C')	4	SPWECBPA	Address of the reply ECB
16(X'10')	80	SPWMSGAR	SPW message area
16(X'10')	64	SPWMSGTX	Message text
80(X'50')	16	SPWREPLY	Operator reply
80(X'50')	1		Reserved
81(X'51')	15	SPWFCBOR SPWREPC1	Start of operator reply
96(X'60')	4	SPWRPECB	Reply ECB for the WTOR
100(X'64')	4	SPWREPID	UCS/FCB ID supplied by the operator reply
104(X'68')	8	SPWFCBIM	Name of the FCB image
104(X'68')	8	SPWUCSIM	Name of the UCS image
	4	SPWPREFX	First half of name
108(X'6C')	4	SPWUCS2H	Last 4 bytes of UCS name
108(X'6C')	4	SPWFCB2H	Last 4 bytes of FCB name

IOB for EXCP Users and OPEN—in User Key

Offset	Length	Name	Description
112(X'70')	40	SPWIOB	IOB area

Information saved from user's IOB

Offset	Length	Name	Description
152(X'98')	4	SPWFLGSV	IOB first word save area
156(X'9C')	4	SPWTRSV	IOBSTART save area
160(X'A0')	4	SPWECBSV	IOB ECB pointer save area

Channel Program Area—in User Key

Offset	Length	Name	Description
168(X'A0')	8	SPWCCW1	First CCW
176(X'A8')	8	SPWCCW2	Second CCW
184(X'B0')	8	SPWCCW3	Third CCW

Work Area for Unpacking Line Numbers—in User Key

Offset	Length	Name	Description
188(X'B8')	4	SPWUNPKA	Unpack area
190(X'BC')	2		Unused bytes
192(X'BE')	2	SPWLNENO	Used by CONVERT instruction

General Work Area—in User Key

Offset	Length	Name	Description
196(X'C0')	4	SPWFFSB	Sense bytes 0-3 from the 3800 Printing Subsystem
200(X'C4')	4	SPWMSGID	Message ID for DOM
204(X'C8')	64	SPWUBLDL	BLDL parameter list for user image library
208(X'CC')	60	SPWULOAD	LOAD parameter list for user image library

BLDL Work Area—SPW5

The BLDL work area is used by the SETPRT executors and is located (when the SETPRT executors are in control) at the address contained in register 8. The BLDL work area is in key 5.

Offset	Length	Name	Description
0(X'0')	64	SPWBLDLA	BLDL list area
0(X'0')	4	SPWBLDLC	Count and length fields
4(X'4')	8	SPWDDNAM	DD name for SYSOUT request
		SPWBLNAM	BLDL name field
7(X'7')	1	SPWFCBQ	FCB name qualifier
		SPWUCSQ	UCS image name qualifier
8(X'8')	4	SPWBLFCB	Load module ID for BLDL
12(X'C')	52		Reserved
64(X'40')	4	SPWWKADR	Address of the SETPRT work area (in user key)

Message Area for the SETPRT Work Area—Key 5

Offset	Length	Name	Description
68(X'44')	8	SPWMSGHS	Header section for message area:
		SPWMLIST	Error message parameter list for IGG08116
76(X'4C')	100	SPWMSGA	SPW message area
		SPWMTXT	Message buffer for IGG08116
176(X'B0')	4	SPWMRECB	Reply ECB for WTOR
180(X'B4')	8	SPWIMAGE	Name of the image requested

3800 Printing Subsystem Area for the SETPRT Work Area—Key 5

Offset	Length	Name	Description
188(X'BC')	4	SPWSPRB	Address of the SVRB extended save area
192(X'C0')	4	SPWSOADD	Address of the SYSOUT work area for IGG08117
		SPWBADDR	Buffer pointer
196(X'C4')	2	SPWSOLEN	Length of the SYSOUT work area
198(X'C6')	2		Reserved
200(X'C8')	4	SPWTWAD1	Address of the translate table work area
		SPWUCSIT	UCS image table pointer
		SPWADFCB	FCB address
204(X'CC')	4	SPWLPTR	FCB line pointer
204(X'CC')	1	SPWTWLN1	Translate table work area subpool number
205(X'CD')	3	SPWTWLN1	Translate table work area length
208(X'D0')	1	SPWFLAG1	Flag byte 1:
	1...	SWPT3800	The SETPRT is for a 3800 Printing Subsystem
	.1..	SPWFCBUA	The FCB is in the user area
	..1.	SPWENDXL	End of DCB exit list
	...1	SPWEXFLD	EXCP for FCB load
	.... 1...	SPWEXWPR	EXCP for writing FCB verify
	.... .1..	SPWENDM	End of section in message area
	.... ..1.	SPWEFCBP	End of FCB verify printout
	1	SPWM128L	Message IEC128D in message area
209(X'D1')	1	SPWFLAG2	Flag byte 2:
	1...	SPWRVMSG	Reissue verify message to 3800 Printing Subsystem
	1..	SPWVMHD	Header section in message area
	..1.	SPWVMCH	'Channel' is in message area
	...1	SPWBLIOB	Build a dummy IOB
	.... 1...	SPWM163L	Message IEC163A is being issued
	.... .1..	SPWM164L	Message IEC164A is being issued
	.... ..1.	SPWNSTOR	SPWN has been stored
	1	SPWNESOI	Not enough space is available to open SYS1.IMAGELIB
210(X'D2')	1	SPWFLAG3	Flag byte 3:
	1...	SPWPLCPY	The SETPRT parameter list has been copied from the user's area to the key 5 work area
	.1..	SPWIMGLD	Image loaded into storage
	..1.	SPWOPNPR	OPEN is processing
	...1	SPWDELRQ	Delete is required
	.... 1...	SPWGRAF0	WCGM 0 has been GCM modified
	.... .1..	SPWGRAF1	WCGM 1 has been GCM modified
	.... ..1.	SPWGRAF2	WCGM 2 has been GCM modified

Offset	Length	Name	Description
	1	SPWGRAF3	WCGM 3 has been GCM modified
211(X'D3')	1	SPWFLAG4	Flag byte 4:
	1...	SPWRPERR	Operator reply error
	.1..	SPWCHKLD	EXCP routine to check for possible lost data
	..1.	SPWNLFCB	No FCB load is required
	...x xxxx		Reserved
212(X'D4')	4	SPWRSNCD	Reason code
212(X'D4')	1	SPWRSN0	Byte 0 of the reason code
213(X'D5')	1	SPWRSN1	Byte 1 of the reason code
214(X'D6')	1	SPWRSN2	Byte 2 of the reason code
215(X'D7')	1	SPWRSN3	Byte 3 of the reason code
216(X'D8')	4	SPWRETCD	Return code
216(X'D8')	1	SPWRET0	Byte 0 of the return code
217(X'D9')	1	SPWRET1	Byte 1 of the return code
218(X'DA')	1	SWPRET2	Byte 2 of the return code
219(X'DB')	1	SPWRET3	Byte 3 of the return code
220(X'DC')	4	SPWIOBST	Address of the IOB standard section
224(X'E0')	2	SPWLNCNT	FCB image line counter
226(X'E2')	2	SPWFCBIL	Length of the FCB image
228(X'E4')	4	SPWCAVTA	Address of the caller's AVT
232(X'E8')	2	SPWWKBTS	Work bytes used to test flags
234(X'EA')	1	SPWI	Total number of translate tables
235(X'EB')	1	SPWJ	Translate table index
236(X'EC')	1	SPWK	Index of the CGM in the translate tables
237(X'ED')	1	SPWL	Work index
238(X'EE')	1	SPWM	Index in the CGM record
239(X'EF')	1	SPWN	Index in the CGM record
240(X'F0')	2	SPWP	Translate table position index
	2	SPWLPISV	Saved previous lines lpi
242(X'F2')	1	SPWMAX	Number of CGMs installed on the printer
243(X'F3')	1	SPWTCBKY	TCB key
244(X'F4')	4	SPWWCGMS	Load WCGM data
		SPWIOREC	Execute control data
248(X'F8')	4	SPWCGMID	Character set IDs
252(X'FC')	20	SPWMEXIT(n	Five 4-byte areas that contain the SETPRT modules' exit list addresses
272(X'110')	1	SPWMEIND	Index for module exit list
273(X'111')	1	SPWERIND	Error index for module exit list
274(X'112')	1	SPWRXIND	Retransmit index for module exit list
275(X'113')	1	SPWAPPOS	Available print positions
276(X'114')	1	SPWPLENG	Length of the SETPRT parameter list
278(X'116')	2		Reserved
280(X'118')	4	SPWADDCB	Address of the caller's DCB
284(X'11C')	4	SPWADDEB	Address of the caller's DEB
288(X'120')	4	SPWADIOB	Address of the IOB prefix section
292(X'124')	4	SPWADUCB	Address of the UCB
296(X'128')	1	SPWCFHIT	FCB half-inch counter
297(X'129')	1	SPWCFB	FCB current index for verify
298(X'12A')	1	SPWFCBO	FCB options
	1...	SPWCPYPS	COPYP specified
	.1..	SPWPSPDS	PSPEED specified
	..xx xxxx		Reserved
299(X'12B')	1		Reserved
300(X'12C')	4	SPWTADDR	Temporary FCB address
304(X'130')	12		Reserved
316(X'13C')	60	SPWSAVE	SETPRT register save area
376(X'178')	72	SPWRSAVE	Compiler register save area
448(X'1C0')	8	SWPFLINU	FCB image line number

Offset	Length	Name	Description
456(X'1C8')	56	SPWSPP	Copy of the user SETPRT parameter list
512(X'200')	1	SPWUKSN	Subpool number of work area in user key
513(X'201')	3	SPWUKLTH	Length of user key work area
516(X'204')	1	SPWK5SN	Subpool number of work area in key 5
517(X'205')	3	SPWK5LTH	Length of key 5 work area
520(X'208')	20	SPWMODWK	Compiler work area for autodata

Error Message Communication Area—User-Provided Area

The error message feedback area is provided by the caller of SETPRT and is pointed to by the SETPRT parameter list field (SPPEMSGA).

Offset	Length	Name	Description
0(X'0')	2	SPWMCLEN	Total length of area
2(X'2')	2	SPWSV02	Reserved
4(X'4')	2	SPWRSV04	Reserved
6(X'6')	2	SPWTXTL	Length of text returned
8(X'8')	2	SPWRSV08	Reserved
10(X'A')	*	SPWTXT	Formatted text (variable length)

SVRB Extended Save Area—Key 0

The SVRB extended save area is used by the SETPRT ESTAE routine.

Offset	Length	Name	Description
0(X'0')	4	SPRMSG	Address of the message CSECT
4(X'4')	4	SPRIDCBA	Address of the image library data set's DCB
8(X'8')	4	SPREXIT	Exit prolog
		SPRREG13	Save area for register 13
12(X'C')	1	SPRKEY	User key
13(X'D')	1	SPRINDIC	Flag byte:
	1...	SPRCNTRL	Control block routine entered
	.1..	SPRTRCNS	GETMAIN unsuccessful
	..1.	SPRUSLIB	User-specified image library
	...1	SPRSYSOT	SETPRT for SYSOUT data set
	 xxxx		Unused
14(X'E')	1	SPRNIOBS	Number of IOBs
15(X'F')	1		Unused
16(X'10')	4	SPRBLDLA	Address of the BLDL list
20(X'14')	4	SPRIOBSV	Address of the IOB altered by IGC0008A
24(X'18')	4	SPRDCBBG	Beginning address of the DCB
28(X'1C')	4	SPRGETMN	Address of the GTRACE buffer
32(X'20')	16	SPRELIST	ESTAE list
32(X'20')	8	SPRTRACE	GTRACE list

3800 Printing Subsystem Translate Table Entry—Key 5

Offset	Length	Name	Description
0(X'0')	288	SPWTT	Translate table, pointed to by SPWTWAD1 (in the SETPRT Work Area)
0(X'0')	8	SPWTTHDR	Translate table header
0(X'0')	4	SPWTTID	Translate table ID
4(X'4')	2		Reserved
6(X'6')	2		Length of character arrangement table
8(X'8')	280	SPWXLAT	Translate table and trailer
8(X'8')	256	SPWTRANS	256 byte translate table
264(X'108')	24	SPWTRAIL	Trailer
264(X'108')	8	SPWTRL1	Four 2-byte entries for character set identifier and loading order
272(X'110')	16	SPWGRAF(n)	Four 4-byte entries for graphic character modification module names

One Entry of an FCB Image for a 3800 Printing Subsystem

Offset	Length	Name	Description
0(X'0')	1	SPWFCBIE	FCB byte for 3800 Printing Subsystem
	00..		Reserved—set to zeros
	..nn	SPWFCBLP	Lines-per-inch bits
	..00		6 lines per inch
	..01		8 lines per inch
	..10		10 lines per inch (Model 3)
	..11		12 lines per inch
	.... nnnn		Channel number 1 to 12, in hexadecimal code

BUFFER POOL CONTROL BLOCK—IGGBCB

Offset	Length	Name	Description
0(X'0')	4	BCBBUFPT	Address of the first buffer (same as BCBBUFAD below)
0(X'0')	1		Filler
1(X'1')	3	BCBBUFAD	Address of first buffer
4(X'4')	1	BCBFLGS	Flag byte
	1...	BCBLRI	Logical record interface present
	.1..	BCBEXTND	BUFCB extended area present
	..xx xxxx		Reserved
5(X'5')	1	BCBBUFNO	Number of buffers
6(X'6')	2	BCBBUFSZ	Size of each buffer
8(X'8')	4	BCBLRIAR	Address of logical record area (same as BCBLRIAD below)
8(X'8')	1		Filler
9(X'9')	3	BCBLRIAD	Address of logical record area
12(X'C')	4	BCBPAD	Padding for doubleword alignment

Offset	Length	Name	Description
	.... 1...	BCBNLN	Length of normal BCB
	.... .1..	BCBEXLN	Length of extension (add to BCBNLN to get total length if BCB is extended)

Users Logical Record Interface Area for SAM Data Sets

Offset	Length	Name	Description
0(X'0')	8	LRILOC	LRI location
0(X'0')	4	LRILGTH	Length of LRI area—LRECL+32 (same as LRILNGTH below)
0(X'0')	1	LRIFLG1	Flags
	1...	LRIEOD	End-of-data reached
	.1..	LRICOB	COBOL data set
	..1.	LRIEOB	EOD after first end-of-block
	...x xxxx		Reserved
1(X'1')	3	LRILNGTH	Length of LRI Area—LRECL+32
4(X'4')	1	LRIFLAG2	Flags
	.1..	LRIRELSE	Release issued
	...1	LRISEG	Segmenting is in process
	.... .1..	LRINTSPN	Nonspanned record
	1	LRIASSEM	Assembling is in process
	x.x. x.x.		Reserved
5(X'5')	1	LRIINDEX	Index to beginning of data
6(X'6')	2	LRIPOS	Position of record in block
8(X'8')	8	LRITRKAD	Track address of beginning record segment
8(X'8')	5	LRIMBBCC	MBBCC of track address (not used if DCB is for output)
13(X'D')	3	LRIRECAD	Record address when record to be written requires segmentation
16(X'10')	4	LRINIOB	Next IOB address (same as LRINXIOB below)
16(X'10')	1		Filler
17(X'11')	3	LRINXIOB	Next IOB address
20(X'14')	2	LRICOUNT	Count field of number of bytes moved
22(X'16')	2		Filler
20(X'14')	4	LRICOUNT	Count field, full word value, of the number of bytes moved when extended logical record interface (XLRI)is used
24(X'18')	8	LRIALIGN	Floating alignment area
24(X'18')	1	LRIDATA	Data

SUBSYSTEM CICB—IGGCICB

Offset	Length	Name	Description
0(X'0')	4	CINXTIOB	Pointer to next IOB; initialized to point to itself
4(X'4')	4	CIECBCD	Psuedo ECB; always marked posted

Offset	Length	Name	Description
8(X'8')	4	CIIOB	Start of basic IOB section
12(X'C')	4	CIECBPTR	Address of pseudo ECB-in prefix
20(X'14')	4	CIRESID	Residual count (CSW); set prior to each SYNAD entry
28(X'1C')	4	CIDCBPTR	DCB address
32(X'20')	56	CIACBD	Data ACB
88(X'58')	20	CIACBED	ACB extension
108(X'6C')	8	CISYNNAM	SYNAD executor name
116(X'74')	4	CIREGSAV	SAM-SI register save area (not used):
120(X'78')	4	CIREGBC	Backward chain pointer (HSA)
124(X'7C')	4	CIREGFC	Foward chain pointer (LSA)
128(X'80')	60	CIREGS	Registers 14 through 12
188(X'BC')	1	CIFLAG1	SYNAD error index
189(X'BD')	1	CIFLAG2	DCB default flags:
	1...	CIFMDSOR	DSORG defaulted in DCB
	.11.	CIFMDEVT	DEVTYPE before OPEN
190(X'BE')	1	CIFLAG3	Control flags:
	1...	CIFFSTP	PUT LOCATE first pass completed
	.1..	CIFCLOSE	CLOSE is processing the DCB
	..1.	CIFRAGM	Record area obtained for BSAM VS
	...1	CIFVSRI	VS record is incomplete
	.... 1...	CIFVSEOB	VBS-format record end-of-block
	.... .1..	CIFERROR	SYNAD is processing an error
	.... ..1.	CIFPOINT	Invalid POINT request
	1	CIFCKBB	SYNAD entry from IGG019BB
191(X'BF')	1	CIFLAG4	Reserved
192(X'C0')	76	CIRPL	Request parameter list (RPL)
268(X'10C')	16	CIRPLEXT	RPL extension
284(X'11C')	20	CIFDBK	RPL feedback area
304(X'130')	4	CILWAREA	Spanned-record work area — BSAM: Size of record area obtained — QSAM: RDW save location

Offset	Length	Name	Description
308(X'134')	4	CISEGLEN	Spanned-record segment length
312(X'138')	4	CIBLKPTR	BSAM spanned-block current address
316(X'13C')	4	CIRAREA	Spanned record area address
320(X'140')	4	CIRECPTR	Spanned record pointer: – BSAM: Current location in record area – QSAM: Spanned segment address
324(X'144')	4	CISAMWA	SAM SI work area
328(X'148')	4	CIWK1	SAM SI work area
332(X'14C')	4	CIWK2	SAM SI work area
336(X'150')	1	CISYNRC	SYNDAF return code
337(X'151')	3	CISYNADA	SYNDAF subroutine address
340(X'154')	1	CIBLDLRC	BLDL return code
341(X'155')	3	CIBLDL	BLDL subroutine address
344(X'158')	1	CIBSPRC	BSP return code
345(X'159')	3	CIBSP	BSP subroutine address
348(X'15C')	1	CIFEOVRC	FEOV return code
349(X'15D')	3	CIFEOV	FEOV subroutine address
352(X'160')	1	CISTOWRC	STOW return code
353(X'161')	3	CISTOW	STOW subroutine address

PARAMETER LIST—IGGPARML

This DSECT expands the parameter list passed to the open/close executors from common open/close.

Offset	Length	Name	Description
0(X'0')	4	PARDCBAD	Address of DCB being opened/closed (same as PARDCBAB below)
0(X'0')	1	PAROPT	OPEN/CLOSE options
	1...	PARENLST	End of list

CLOSE Options

Offset	Length	Name	Description
	.1..	PARREWND	REWIND

Offset	Length	Name	Description
	..11	PARLEAVE	LEAVE
	..1.	PARFREE	Unallocate during CLOSE
	...1	PARRREAD	REREAD

OPEN Options

Offset	Length	Name	Description
	 1111	PAROUTPT	Output
	.... .111	PAROUTIN	Outin
	.... .1..	PARUPDAT	Update
	11	PARINOUT	Inout
	1	PARRDBCK	Readback
	....	PARINPUT	Input
1(X'1')	3	PARDCBAB	Address of DCB being open/closed

PRINTER DEVICE CHARACTERISTICS TABLE—IGGPDC

A printer device characteristics table (PDCT) is generated during system generation for each 3203-4, 3203-5, 3211, 3262 Model 5, 4245, and 4248 printer. The PDCT is appended to the unit record UCB UCS extension.

Offset	Length	Name	Description
0(X'0')	2	PDCMPLEN	Maximum print line length
2(X'2')	2	PDCUCSL	UCS length
4(X'4')	2	PDCERPWL	OBR/MDR work area length
6(X'6')	1	PDCDCBC	DCB device type code
7(X'7')	1	PDCFCBP	FCB prefix id
8(X'8')	1	PDCUCSP	UCS prefix id
9(X'9')	1	PDCFLAG1	Flag byte 1
	1...	PDCUCSIT	Device uses UCS image table
	.1..	PDCLDPOS	Lost data condition possible
	..1.	PDCAUTOP	Hardware positions paper when FCB loaded
	...1	PDCPTDEV	Page tracking device
	.... 1...	PDCFCBOP	FCB may contain options
	.... .1..	PDCCLRPT	Clear printer command supported
	.... ..1.	PDCLRUCS	Limited function read UCSB
	1	PDCSIGAT	Signal attention command supported
10(X'A')	1	PDCFLAG2	Flag byte 2—Reserved
11(X'B')	1	PDCAFCBP	Alternate FCB prefix id
12(X'C')	4		Reserved

If UCBUCSE=YES is specified on the IGGPDC macro the following mapping is provided for the UCB UCS extension

Offset	Length	Name	Description
0(X'0')	16	PDCUCS	UCB UCS extension
16(X'10')	2	PDCSPGID	Synchronization page id
18(X'12')	2	PDCPDCTO	Offset to PDCT

SAM OPEN/CLOSE WORK AREA—IGGSCW

This DSECT maps against the O/C/EOV work area fields DXCCW1 through DXCCW12. The purpose of the DSECT is to give meaningful equates to these fields when used by the open and close executors. The comments for each label indicate whether the field is used by the OPEN executors (-O) or the CLOSE executors (-C).

Offset	Length	Name	Description
368(X'170')	8	DXCCW1	
368(X'170')	4	DXBLDL	BLDL parameter list -O
368(X'170')	4	SCWGETMA	Register save area for QSAM routines -C
368(X'170')	1	DXCCWOP	CCW OP code -O
368(X'170')	1	SCWSAVCD	Problem determination code -C
368(X'170')	1	DXUCSUCB	UCB UCS options -O
369(X'171')	3	DXCCWADR	Buffer address -O
369(X'171')	3	SCWGETMB	(Same as SCWGETMA) -C
372(X'174')	4	DXBLDLIM	Image name for BLDL -O
372(X'174')	2	DXCCWFLG	CCW flags -O
374(X'176')	2	DXCCWBYT	CCW byte count -O
376(X'178')	8	DXCCW2	
376(X'178')	4	DXIMGNAM	Image name -O
376(X'178')	4	SCWRALL	Save area for all registers -C
408(X'198')	8	DXCCW6	
408(X'198')	8	DXSAVUCS	Area to save UCS name -O
416(X'1A0')	8	DXCCW8	
416(X'1A0')	4	DXIMGDCB	Address of SYS1.IMAGELIB DCB -O
424(X'1A8')	8	DXCCW9	
424(X'1A8')	8	DXSAVFCB	Area to save FCB name -O
432(X'1B0')	8	DXCCW10	
432(X'1B0')	16	DXFCBUCS	UCS and FCB parameter fields -O
432(X'1B0')	8	DXFCBP	To clear FCB parameter field -O
432(X'1B0')	1	DXFCBSW1	Switch for FCB parameters -O
433(X'1B9')	1	DXABEND	Indicates DMABCOND to be issued -O
433(X'1B9')	1	DXFLAG1	FCB flag byte -O
434(X'1BA')	1	DXSTAGE2	Indicates next executor is stage 2 executor -O
437(X'1B5')	1	DXFCBOPT	JFCB FCB options -O
438(X'1B6')	4	DXFCBID	FCB image identification -O
442(X'1BA')	8	DXCCW11	
442(X'1BA')	8	DXUCSP	Parameter list for UCS -O
442(X'1BA')	1	DXUCSSW1	Switch for UCS parameters -O
442(X'1BA')	1	DXABRETC	Internal return code for problem determination -O
443(X'1BB')	1	DXEROPT	To save DCBEROPT -O
444(X'1BC')	1	DXNABEND	Indicate ABEND in control -O
445(X'1BD')	1	DXUCSOPT	JFCB UCS options -O
446(X'1BE')	4	DXUCSID	UCS image identification -O
450(X'1C2')	8	DXCCW12	
450(X'1C2')	4	SCWXCTLP	Supervisor parameter list for XCTL -O/C

SAM/PAM/DAM GTRACE BUFFER—IGGSPD

Offset	Length	Name	Description
0(X'0')	[1]	SPDBFR	SPD buffer input record
0(X'0')	9	SPDHDR	Buffer header
0(X'0')	8	SPDDDNAM	DD name from TIOT
8(X'8')	1	SPDABCCD	ABEND condition code
9(X'9')	[1]	SPDTRACE	Trace record area (variable - maximum length is 247 bytes)
9(X'9')	[1]	SPDTRRCD	Trace record
9(X'9')	2	SPDRCDHD	Trace record header
9(X'9')	1	SPDRCDLN	Trace record length

[1] Depends on the length of the block to be traced (maximum block length is 245 bytes including the block address, if present).

Offset	Length	Name	Description
10(X'A')	1	SPDBLKID	ID of trace record
11(X'B')	[1]	SPDDATA1	Block to be traced (no block addresspresent)
11(X'B')	4	SPDBLKAD	Address of block to be traced (not present for block with ID less than 127)
15(X'F')	[1]	SPDDATA2	Block to be traced (block address present)

[1] Depends on the length of the block to be traced (maximum block length is 245 bytes including the block address, if present).

STOW WORK AREA—IGGSTW

This DSECT maps the work area used by the STOW modules.

Offset	Length	Name	Description
0(X'0')	4	STWPARM	Address of user-supplied entry name (lower of two for change)
4(X'4')	4	STWHIGH	Address of higher of two user-supplied names (change only)
8(X'8')	2		Reserved
10(X'A')	2	STWOFFLW	Offset to add, replace, or delete location in low block
12(X'C')	8	STWOLDNM	Name of entry being deleted
20(X'14')	8	STWNEWNM	Name of new entry
28(X'1C')	3	STWTTR	Relative address of member
31(X'1F')	1	STWCTTRN	Alias bit, number of TTRNs, and length of user data
	1...	STWALIAS	This member name is an alias
32(X'20')	62	STWDATA	User data for entry

Flag, Condition, and Switch Bytes

Offset	Length	Name	Description
94(X'5E')	1	STWFLAG1	First flag byte bit definitions
	1...	STWCHNG	Change function (used in combination with STWADD and STWDEL)
	.1..	STWDEL	Delete function
	..1.	STWREPL	Replace function
	...1	STWADD	Add function
	.... 1...	STWDRYRN	Dry run being made on directory
	.... .1..	STWFLOW	Used to control program flow
	.... ..1.	STWDCBWR	Last DCB operation was a WRITE
	1		Reserved
95(X'5F')	1	STWRTN	Return code save area

Control Blocks for STOW Channel Programs

Offset	Length	Name	Description
96(X'60')	4		If already on a doubleword; go to the next fullword boundry

Event Control Block

Offset	Length	Name	Description
100(X'64')	4	ECBRB	RB (request block) address while waiting for event
100(X'64')	1	ECBCC	Completion code byte
	1...	ECBWAIT	Waiting for completion of event
	.1..	ECBPOST	Event completed
	.111 1111	ECBNORM	Channel program terminated without error
	.1.. ...1	ECBPERR	Channel program terminated with permanent errors or for BTAM completed with an I/O error
	.1.. ..1.	ECBDAEA	Channel program terminated because a direct access extent address was violated
	.1.. ..11	ECBABEND	I/O abend condition occurred for error transient loading task
	.1.. .1..	ECBINCPT	Channel program intercepted because of permanent error associated with device end for previous request; the intercepted request can be reinitiated.
	.1.. 1...	ECBREPRG	Request element for channel program made available after it has been purged
	.1.. 1...	ECBHALT	Enable command halted
	.1.. 1.11	ECBERPAB	Abnormal completion of processing because of a critical error such as the presence of invalid control block fields
	.1.. 1111	ECBERPER	Error recovery routines entered because of direct access error are unable to read home address of record 0

Offset	Length	Name	Description
101(X'65')	3	ECBRBA	Request block address (while awaiting completion of an event)
101(X'65')	3	ECBCCCNT	Zeros or remainder of completion code (after completion of the event)

Prefix Sections of the IOB

Offset	Length	Name	Description
88(X'58')	8	IOBPREFX	Prefix sections
88(X'58')	8	IOBQSAMC	QSAM/BSAM/BPAM prefix
88(X'58')	8	IOBBSAMC	Chained scheduling
88(X'58')	8	IOBBPAMC	16 bytes
88(X'58')	1	IOBCFLG1	Flag byte
	1....	IOBRSV01	Reserved
	.1..	IOBRSV02	Reserved
	..1.	IOBRSV03	Reserved
	...1	IOBRSV04	Reserved
	.... 1...	IOBPTST	NOTE or POINT operation in process
	.... .1..	IOBABAPP	Error processed once by abnormal-end appendage
	.... ..1.	IOBRSTCH	Restart channel
	1	IOBPCI	PCI interrupt has occurred
89(X'59')	1	IOBRSV05	Reserved
90(X'5A')	1	IOBCINOP	Offset of last I/O command for input operation (NOP CCW) from the ICB origin
91(X'5B')	1	IOBCONOP	Offset of last I/O command for output operation (NOP CCW) from the ICB origin
92(X'5C')	4	IOBCECB	Event control block
96(X'60')	4	IOBCICB	Address of first ICB on queue
100(X'64')	4	IOBCNOPA	Address of NOP command at end of queue

QSAM BSAM BPAM Prefix

Offset	Length	Name	Description
96(X'60')	8	IOBQSAMN	QSAM/BSAM/BPAM prefix
96(X'60')	8	IOBBSAMN	Normal scheduling
96(X'60')	8	IOBBPAMN	8 bytes
96(X'60')	4	IOBNIOBA	Address of next IOB on chain
96(X'60')	1	IOBNFLG1	Flag byte
	1...	IOBPRTOV	PRTOV occurred
	.1..	IOBWRITE	WRITE operation in process
	..1.	IOBREAD	READ operation in process
	...1	IOBUPDAT	Block is to be updated
	.... 1...	IOBBKSPC	IOB is being used for BSP CTRL NOTE/POINT
	.... .1..	IOBSPAN	Spanned record
	.... ..1.	IOBUPERR	Update channel program has been split
	1	IOBFIRST	First IOB on chain
97(X'61')	3	IOBNIOBB	Address of the next IOB on the chain
100(X'64')	4	IOBNECB	Event control block address

Standard Section of the IOB

Offset	Length	Name	Description
104(X'68')	8	IOBSTDRD	
104(X'68')	1	IOBFLAG1	Flag byte
	1...	IOBDATCH	Data chaining used in channel program
	.1..	IOBCMDCH	Command chaining used in channel program
	..1.	IOBERRTN	Error routine is in control
	...1	IOBRPSTN	Device is to be repositioned
	.... 1...	IOBCYCCK	Cyclic r
	.... 1...	IOBFCREX	FETCH command retry exit (direct access only)
	.... .1..	IOBIOERR	I/O error has occurred
	.... ..1.	IOBUNREL	I/O request is unrelated (nonsequential)
	1	IOBSPSVC	SAM/PAM flag set by SVC if I/O appendage should not process interrupt
105(X'69')	1	IOBFLAG2	Flag byte
	1...	IOBHALT	HALT I/O issued by SVC PURGE routine
	.1..	IOBSENSE	Issue SENSE command after device end occurs
	..1.	IOBPURGE	IOB purged—allow I/O to quiesce
	...1	IOBRDHA0	Home address to be read—no seek needed
	.... 1...	IOBALTTR	No test for out-of-extent—alternate track in use
	.... .1..	IOBSKUPD	Seek address is being updated—cylinder end or file mask violation has occurred
	.... ..1.	IOBSTAT0	Device end status OR-ed with channel end status—graphics device
	1	IOBPNCH	Turned on by QSAM when error recovery is to be provided for the 2540 Card Punch
106(X'6A')	1	IOBSENS0	First sense byte
	1...	IOBS0B0	Bit 0 (device dependent)
	.1..	IOBS0B1	Bit 1 (device dependent)
	..1.	IOBS0B2	Bit 2 (device dependent)

Offset	Length	Name	Description
	...1	IOBS0B3	Bit 3 (device dependent)
	.... 1...	IOBS0B4	Bit 4 (device dependent)
	.... .1..	IOBS0B5	Bit 5 (device dependent)
	.... ..1.	IOBS0B6	Bit 6 (device dependent)
	1	IOBS0B7	Bit 7 (device dependent)
	1	IOBSNSC9	Channel 9 sensed in carriage tape
107(X'6B')	1	IOBSENS1	Second sense byte
	1...	IOBS1B0	Bit 0 (device dependent)
	.1..	IOBS1B1	Bit 1 (device dependent)
	..1.	IOBS1B2	Bit 2 (device dependent)
	...1	IOBS1B3	Bit 3 (device dependent)
	.... 1...	IOBS1B4	Bit 4 (device dependent)
	.... .1..	IOBS1B5	Bit 5 (device dependent)
	.... ..1.	IOBS1B6	Bit 6 (device dependent)
	1	IOBS1B7	Bit 7 (device dependent)
108(X'6C')	4	IOBECBPT	Address of ECB to be posted upon completion of I/O
108(X'6C')	1	IOBECBCC	Completion code for current I/O request
109(X'6D')	3	IOBECBPB	Address of ECB to be posted upon completion of I/O
112(X'70')	1	IOBFLAG3	Error routine flag byte
	1...	IOBCCC	Channel control check error count
	.1..	IOBICC	Interface control check error count
	..1.	IOBCDC	Channel data check error
	...1	IOBACU	Attention/control unit error
	.... 1...	IOBCNC	Chain check error
	.... .1..	IOBMSG	Message flag
	.... ..1.	IOBICL	Incorrect length error
	1	IOBLOG	Log-out flag
113(X'71')	7	IOBCSW	Seven low-order bytes of CSW at channel end
113(X'71')	3	IOBCMDA	Command address (3890)
116(X'74')	2	IOBSTBYT	Status bits 32-47 (3890)

Offset	Length	Name	Description
118(X'76')	2		Last two bytes of IOBCSW
120(X'78')	4	IOBSTART	Address of channel program
120(X'78')	1	IOBSIOCC	Bits 2 and 3 = condition code from SIO
121(X'79')	3	IOBSTRTB	Address of channel program
124(X'7C')	4	IOBDCBPT	Address of data control block for this IOB
124(X'7C')	1	IOBFLAG4	Flag byte
	1...	IOBGDPOL	Reenter SIO appendage for OLTEP guaranteed device path
	.1..	IOBRSV38	Reserved
	..1.	IOBRSV39	Reserved
	...1	IOBRSV40	Reserved
	.... 1...	IOBRSV41	Reserved
	.... .1..	IOBRSV42	Reserved
	.... ..1.	IOBRSV43	Reserved
	1	IOBRSV44	Reserved
125(X'7D')	3	IOBDCBPB	Address of data control block for this IOB
128(X'80')	4	IOBRESTR	Restart address for error retry
128(X'80')	1	IOBREPOS	Code used to reposition device
129(X'81')	3	IOBRSTRB	Restart address for error retry
132(X'84')	2	IOBINCAM	Value used to increase block count on tape
132(X'84')	1	IOBCRDCC	Optical reader—data check error count
133(X'85')	1	IOBCRILC	Optical reader—Incorrect Length error count
134(X'86')	2	IOBERRCT	Count of error retries

Direct Access Extension Section of the IOB

Offset	Length	Name	Description
136(X'88')	1	IOBM	Relative extent number for this request (0-15)
137(X'89')	2	IOBBD	Bin number (data cell)
137(X'89')	1	IOBBB1	

Offset	Length	Name	Description
138(X'8A')	1	IOBBB2	
139(X'8B')	2	IOBCC	Cylinder number
139(X'8B')	1	IOBCC1	
140(X'8C')	1	IOBCC2	
141(X'8D')	2	IOBHH	Track number
141(X'8D')	1	IOBHH1	
142(X'8E')	1	IOBHH2	
143(X'8F')	1	IOBR	Record number

STOW Channel Programs

Offset	Length	Name	Description
144(X'90')	8	STWINCP	Channel program to read the initial 4 two directory blocks
144(X'90')	8	STWSRCH1	Search ID equal
152(X'98')	8	STWTIC11	Transfer control to search ID
160(X'A0')	8	STWRDCT1	Read count
168(X'A8')	1	STWSRKY1	Search on key equal or high
169(X'A9')	3	STWKYAD1	Key address
172(X'AC')	4		Flags and byte count
176(X'B0')	8	STWTIC12	Transfer control to read count
184(X'B8')	8	STWRDAT1	READ data
192(X'C0')	8	STWRCKD1	READ count key data
200(X'C8')	8	STWWRDCP	Channel program to WRITE and READ directory blocks
200(X'C8')	1	STWSRCH2	Search ID equal
201(X'C9')	3	STWIDAD2	ID address
204(X'CC')	4		Flags and byte count
208(X'D0')	8	STWTIC2	Transfer control to search ID
216(X'D8')	1	STWWRKD1	WRITE key and data
217(X'D9')	3	STWWRAD2	WRITE address
220(X'DC')	1	STWWFLG2	Flags
	.1..	STWCMDCH	Command chain to next CCW

Offset	Length	Name	Description
221(X'DD')	3		Byte count
224(X'E0')	1	STWRCKD2	READ count key data
225(X'E1')	3	STWRDAD2	READ Address
228(X'E4')	4		Flags and byte count
232(X'E8')	1	STWRCKD3	READ count key data
233(X'E9')	3	STWRDAD3	READ address
236(X'EC')	4		Flags and byte count

STOW Input/Output Buffers

(For details, see Buffer DSECT, below.)

Offset	Length	Name	Description
240(X'F0')	276	STWBUF1	Initially contains the first of two directory blocks read
516(X'204')	276	STWBUF2	Initially contains the second of two directory blocks read
792(X'318')	276	STWBUF3	Initially used as the first output buffer
1072(X'430')	8	STWEND	End of work area

Map of STOW Input/Output Buffers

Offset	Length	Name	Description
0(X'0')	8	BUFCNT	Count field containing absolute disk address
0(X'0')	5	BUFCCHHR	CCHHR field
5(X'5')	3	BUFKDD	Key and data length
8(X'8')	8	BUFKEY	Key field (highest member name)
16(X'10')	256	BUFDATA	Data Area
16(X'10')	2	BUFN	Number of bytes used in this directory block
18(X'12')	254	BUFENTRY	Directory entries
272(X'110')	4	BUFADDR	Used to chain buffers

SVRB Extended Save Area

Offset	Length	Name	Description
0(X'0')	4	XSAREG4	Save area for register 14
4(X'4')	4	XSASTWWA	Address of STOW work area
8(X'8')	16	XSAESTAE	List form of the ESTAE macro instruction.

SYNADAF GENERAL REGISTERS SAVE AREA AND MESSAGE BUFFER AREA—IGGSYN

Offset	Length	Name	Description
0(X'0')	72	SYNSAVE	Save area
0(X'0')	4	SYNPL1	Used by PL/I language program
4(X'4')	4	SYNPREV	Address of previous save area
8(X'8')	4	SYNNEXT	Address of next save area
12(X'C')	60	SYNGRS	General register save area
72(X'48')	8	SYNVLFLD	Length field for variable-length records
80(X'50')	1	SYNMSG	Data Area
80(X'50')	6	SYNREAD	Data area for READ
80(X'50')	4	SYNRDERR	Return information if read error
84(X'54')	2	SYNBYTRD	Number of bytes read
86(X'56')	35	SYNWAREA	Work area
86(X'56')	1	SYNPURG	Error type indicator
86(X'56')	1	SYNACMTH	Access method input code
87(X'57')	1		Unused
88(X'58')	8	SYNWRKA	Work Area
88(X'58')	4	SYNWKA1	Work area number 1
92(X'5C')	4	SYNWKA2	Work area number 2
96(X'60')	4		Unused
100(X'64')	20	SYNWORK	Work area
120(X'78')	1	SYNSTART	Blank
121(X'79')	1	SYNCMMA1	Comma
122(X'7A')	8	SYNJOBNM	Job name
130(X'82')	1	SYNCMMA2	Comma

Offset	Length	Name	Description
131(X'83')	8	SYNSTPNM	Step name
139(X'8B')	1	SYNCMMA3	Comma
140(X'8C')	3	SYNUNTID	Unit address
143(X'8F')	1	SYNCMMA4	Comma
144(X'90')	2	SYNDVTYP	Device type
146(X'92')	1	SYNCMMA5	Comma
147(X'93')	8	SYNDDNM	DD name
155(X'9B')	1	SYNCMMA6	Comma
156(X'9C')	6	SYNOPRTN	Operation attempted
162(X'A2')	1	SYNCMMA7	Comma
163(X'A3')	15	SYNERROR	Error description
178(X'B2')	1	SYNCMMA8	Comma
179(X'B3')	14	SYNPOS	Area to unpack ICB seek address
179(X'B3')	7	SYNPOSM1	Unused—magnetic tape
186(X'BA')	7	SYNPOSM2	Area to unpack block count for magnetic tape
186(X'BA')	6	SYNPOSMV	Unpack value
192(X'C0')	1	SYNPOSMS	Sign byte in unpack format
193(X'C1')	1	SYNCMMA9	Comma
193(X'C1')	5	SYNACCSS	Access method type
199(X'C7')	1	SYNBLNK2	Blank
200(X'C8')	4	SYNPRMR1	Parameter register save area
204(X'CC')	4	SYNPRMR2	Parameter register save area
208(X'D0')	4	SYNEND	End of IGGSYN

SYNADAF and SYNADRLS SVRB Extended Save Area

Offset	Length	Name	Description
160(X'A0')	4	SYNRETA	Return address
164(X'A4')	16	SYNXCTPL	XCTL parameter list
164(X'A4')	4	SYNXCTEP	Address of the entry point name
168(X'A8')	12	SYNXCTLT	List of parameters
168(X'A8')	4	SYNXCTDB	Address of the DCB
172(X'AC')	8	SYNXCTNM	Entry point name

Offset	Length	Name	Description
176(X'B0')	1	SYNXCTID	Load module ID
180(X'B4')	1	SYNESTPL	Flags for TCB PURGE and ASYNCH
181(X'B5')	3		Exit address not specified
184(X'B8')	4		Parameter list address not specified
188(X'BC')	4		TCB not specified
192(X'C0')	1		Flags
193(X'C1')	3		Reserved
196(X'C4')	4	SYNESFLG	ESTAE routine flag word
	1...	SYNGTM	Return from GETMAIN without error
	.1..	SYNCSA	Save areas chained successfully
	..1.	SYNMLC	Message CSECT loaded
	...1	SYNRCS	Caller's save area restored successfully
	.... 1...	SYNESTAE	ESTAE routine entered
	xxx		Reserved
198(X'C6')	1	SYNURKEY	User key
199(X'C7')	9	SYNUNPKA	Work area for unpack

SETPRT PARAMETER LIST—IHASPP

Offset	Length	Name	Description
0(X'0')	1		Unused
1(X'1')	3	SPPDCBB	DCB address
4(X'4')	4	SPPUCS	UCS module ID (not used for 3800 Printing Subsystem)
8(X'8')	1	SPPLDMOD	Flag byte describing load mode (not used for 3800 Printing Subsystem)
	.0..		UCS= fold not specified
	.1..		UCS= fold
	x.xx xxxx		Reserved
9(X'9')	1	SPPVERFY	Flag byte describing UCS verify (not used for 3800 Printing Subsystem)

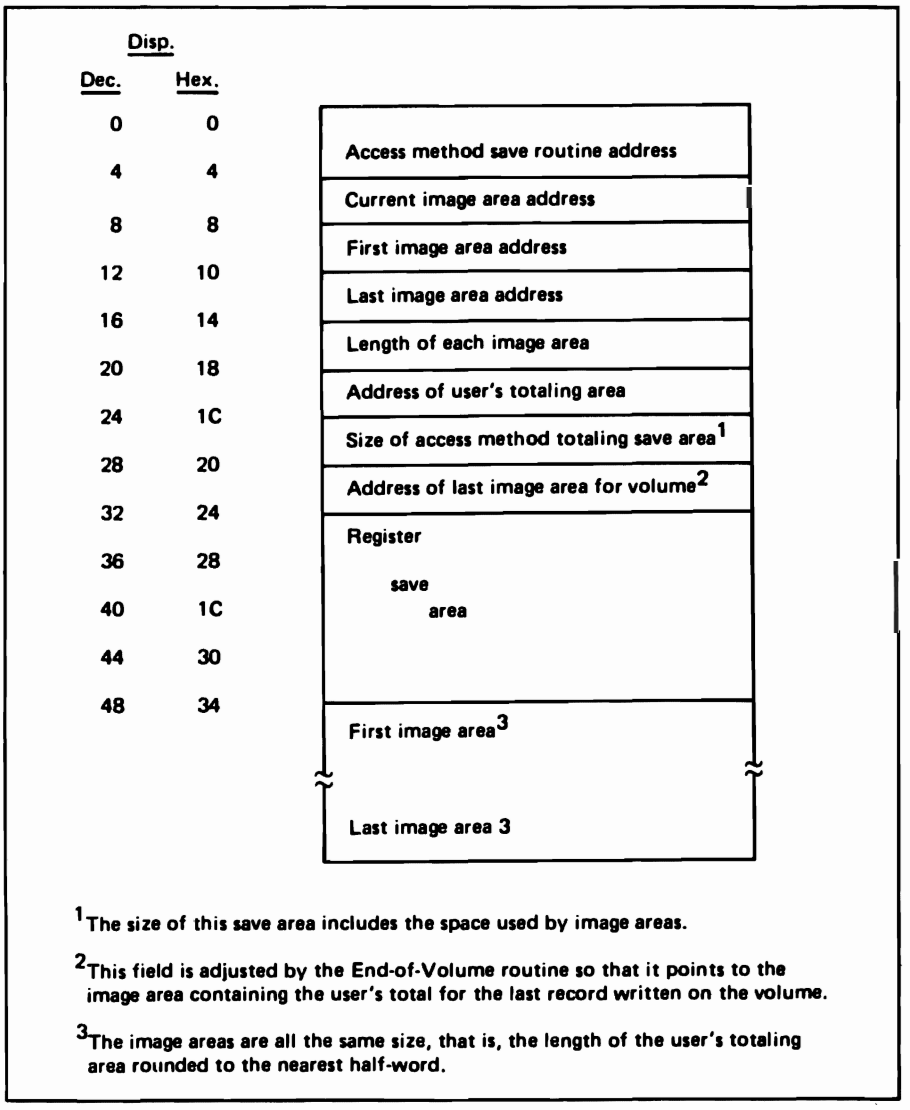
Offset	Length	Name	Description
	...0		No verification for UCS
	...1		UCS verification requested
	xxx. xxxx		Unused
10(X'A')	1	SPPFDUNF	Flag byte:
	10..	SPPFBLK	Block data checks
	01..	SPPFUBLK	Unblock data checks
	..10	SPPSCHED	Schedule SYSOUT data set segment for printing now
	..01	SPPNOSCD	Do not schedule SYSOUT segment for immediate printing
	.... 10..	SPPUNFLD	OPTCD= unfold option
	.... 01..	SPPFOLD	OPTCD= fold option
	.... ..x.		Unused
	1	SPPEXTL	SETPRT parameter list is at least 48 bytes long
11(X'B')	4	SPP FCB	FCB module ID or address of in-storage FCB module (see SPPFLAG2)
15(X'F')	1	SPPVERAL	FCB flag byte:
	0...		No FCB image verification requested
	1...		Print FCB image for verification
	.xxx xxx.		Unused
	0		No forms alignment requested (not used for 3800 Printing Subsystem)
	1		Issue WTOR for forms alignment (not used for 3800 Printing Subsystem)
16(X'10')	1	SPPFLAG1	Flag byte number 1:
	0...	SPPBURST	Thread continuous forms stack
	.1..	SPPBURST	Thread burster-trimmer-stacker
	.1..	SPPREX	Retransmission - only change COPIES, FLASH, and starting copy number
	..1.	SPPINIT	Issue initialize printer CCW
	...1	SPPNOMSG	Suppress error messages on the printer
	.... 1...	SPPBFREQ	Bypass forms overlay WTOR or bypass band mount message
	.... .1..	SPPBTREQ	Bypass threading change WTOR

Offset	Length	Name	Description
	.... ..1.	SPPBOMSG	Bypass WCGM's exceeded error message
	1	SPPFORC	JES force load of the FCB
17(X'11')	1	SPPFLAG2	Flag byte number 2:
	0...		Copy modification, if specified, is a module ID.
	1...		Copy modification is specified as an address.
	.0..		Character arrangement table 0 if specified is a module ID.
	.1..		Character arrangement table 0 is specified as an address.
	..0.		Character arrangement table 1 if specified is a module ID.
	..1.		Character arrangement table 1 is specified as an address.
	...0		Character arrangement table 2 if specified is a module ID.
	...1		Character arrangement table 2 is specified as an address.
	.... 0...		Character arrangement table 3 if specified is a module ID.
	.... 1...		Character arrangement table 3 is specified as an address.
	.... .0..		FCB is specified as a module ID.
	.... .1..		FCB is specified as an address.
	xx		Unused bits.
18(X'12')	1	SPPCPYNR	The number of copies to be printed.
19(X'13')	1	SPPSTCNR	The copy number of the first copy to be printed.
20(X'14')	2	SPPLEN	Length of the parameter list.
22(X'16')	1	SPPFRMNR	The number of copies to be forms flashed, starting with the first printed copy.
23(X'17')	1	SPPTRC	The character arrangement table to be selected when the specified copy modification record is loaded in the printer.
24(X'18')	4	SPPMODPT	The module ID or in-storage address of the copy modification module.
28(X'1C')	4	SPPIMAGE	The identifier for the forms overlay frame.

Offset	Length	Name	Description
32(X'20')	4	SPPXLAT1	The module ID or in-storage address of character arrangement table module 0.
36(X'24')	4	SPPXLAT2	The module ID or in-storage address of character arrangement table module 1.
40(X'28')	4	SPPXLAT3	The module ID or in-storage address of character arrangement table module 2.
44(X'2C')	4	SPPXLAT4	The module ID or in-storage address of character arrangement table module 3.
48(X'30')	4	SPPEMSGA	Address of the message communication area for error information.
52(X'34')	4	SPPLIDCB	Address of the library DCB for 3800 Printing Subsystem load modules.
56(X'38')	1	SPPCOPYP	COPYP specification
57(X'39')	1	SPPFLAG3	Flag byte number 3:
	1...	SPPCPYPS	COPYP specified
	.1..	SPPPSPDS	PSPEED specified
	..11		Reserved
	 1111		Caller's PSPEED specification as follows:
	.... 00..		Unchanged
	.... 01..		Low
	.... 10..		Medium
	.... 11..		High
	00		Reserved; must be zero
58(X'3A')	6		Reserved

ACCESS METHOD SAVE AREA FOR USER TOTALING

The access method save area for user totaling is pointed to by the address in bytes 5 through 7 in the EXCP access method, BSAM, or QSAM-dependent section of the DEB. See Figure 34 on page 241.



[1]The size of this save area includes the space used by image areas.

[2]This field is adjusted by the End-of-Volume routine so that it points to the image area containing the user's total for the last record written on the volume.

[3]The image areas are all the same size, that is, the length of the user's totaling area rounded to the nearest half-word.

Figure 34. Access Method Save Area for User Totaling

DIAGNOSTIC AIDS

OPEN AND CLOSE EXECUTOR PROBLEM DETERMINATION

For information on tracing module and data flow during execution of the OPEN and CLOSE executors that issue the IECRES macro, see "Problem Determination" in Open/Close/EOV Logic. Note that the access method executors do not have transfer control tables as common OPEN and CLOSE modules do.

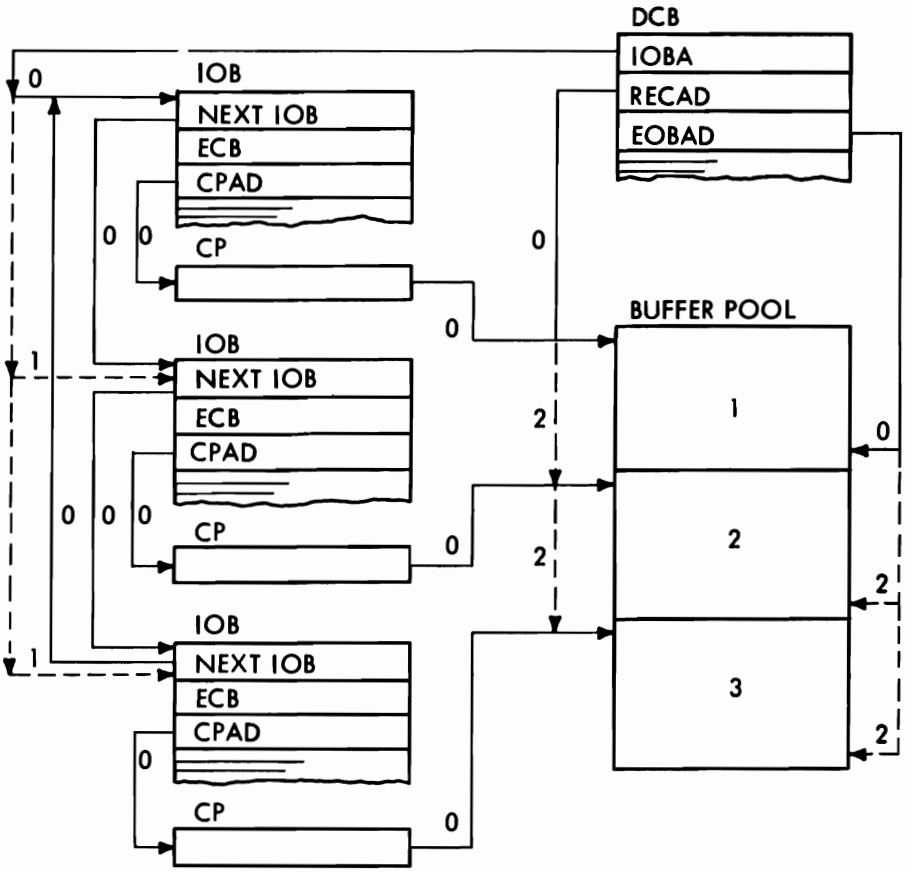
QSAM CONTROL BLOCKS

Figure 35 on page 243 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.

BSAM CONTROL BLOCKS

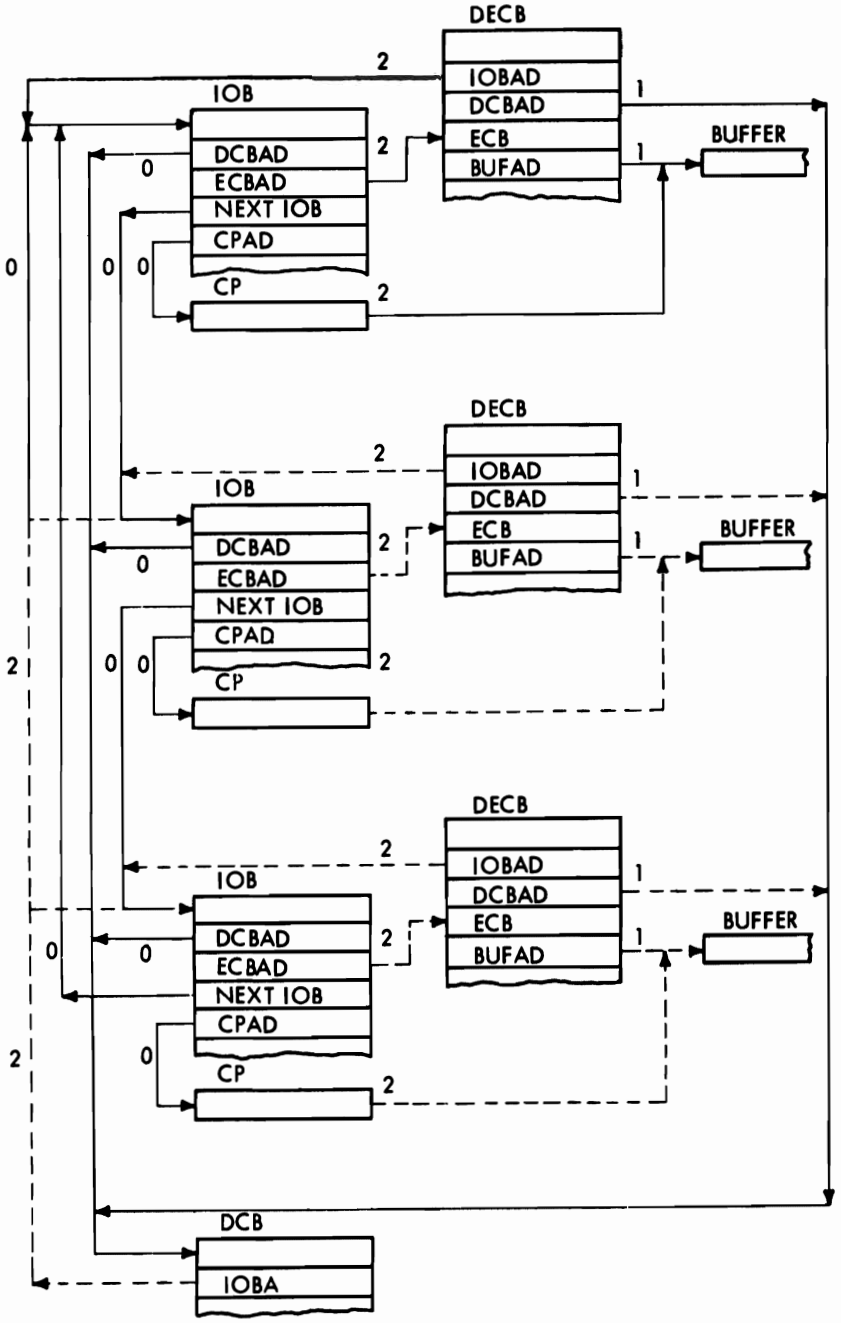
Figure 36 on page 244 shows the control blocks used in BSAM and their stages of completion. Stage 0 shows the state of the control blocks before any READ or WRITE macro instruction. Stage 1 shows the effect of the READ or WRITE macro instruction, that is, the values supplied by the processing program in the data event control block (DECB). Finally, stage 2 shows the effect of the READ or WRITE routine having tied 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 program (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 Updated by the synchronizing routine.
2 Updated by the GET or PUT routine.
– – – Successive Address Values

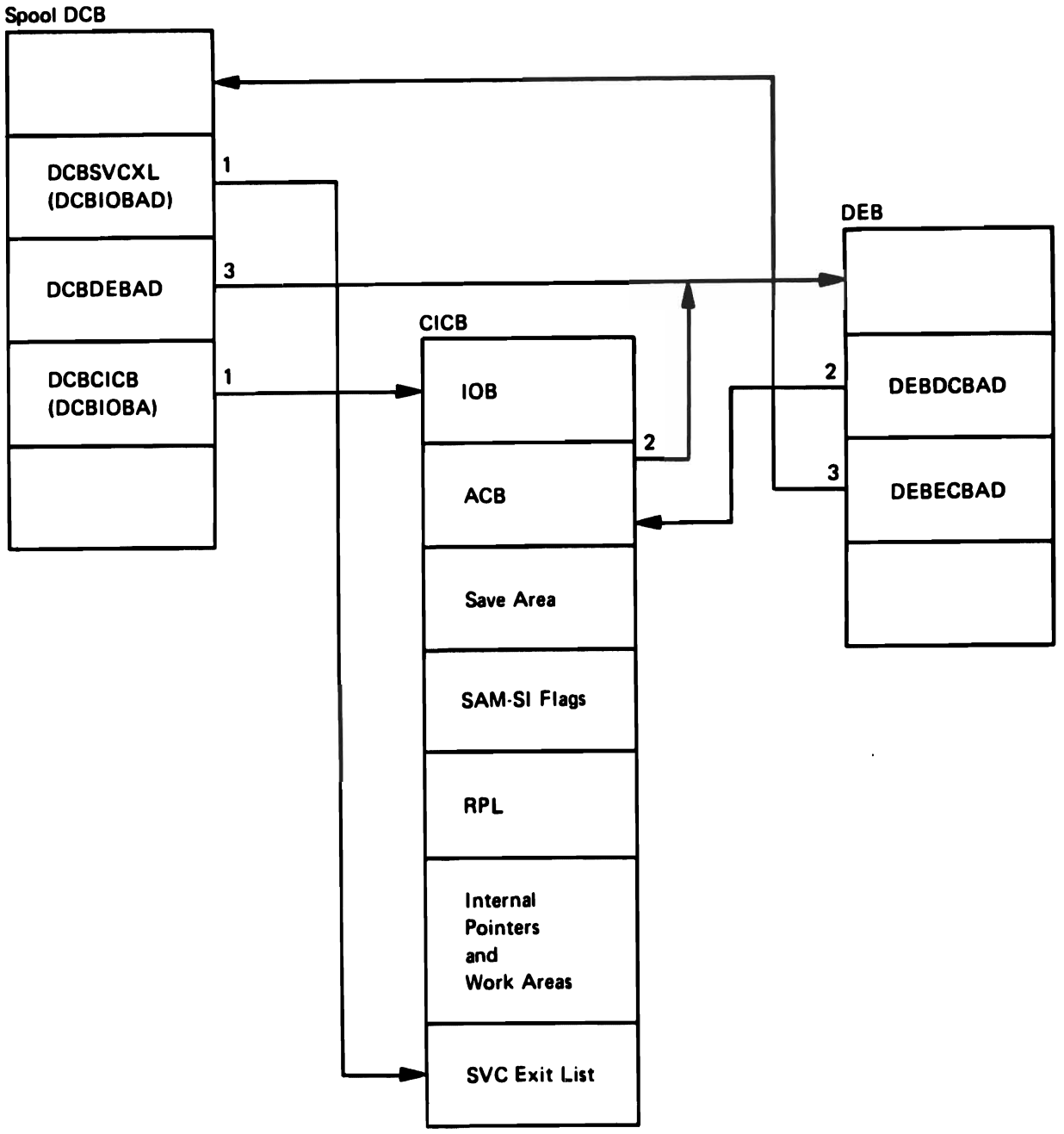
Figure 35. QSAM Control Blocks



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 36. BSAM Control Blocks

Figure 37 shows the relationship of the SYSIN/SYSOUT data sets.



Notes:
1. Set by DCB OPEN executors (SAM-SI).
2. Set by ACB OPEN (JES2 and JES3 open executors).
3. Set by DCB OPEN executors after ACB open.

Figure 37. Control Block Structure for SYSIN/SYSOUT Data Sets

ABEND CODES AND CROSS-REFERENCE TABLE

This table can be used to determine which module detected an abnormal termination condition. The system code and return code (register 15) are used to identify the message ID and the module name and can be found in dumps and in the message text. For example, in the message

IEC141I, 013-BC, IGG0199G, JJOUFPN5,,SYSIN

013 is the system code and BC is the return code. Information in the table should be used in conjunction with System Messages and System Codes.

Abend Code	Return Code	Module	Message Number
001		IGG019AH	
		IGG019AN	
002	04	IGG019AB	IEC036I
		IGG019AD	
		IGG019AE	
		IGG019BN	
		IGG019BO	
		IGG019FB	
		IGG019FD	
		IGG019FF	
	08	IGG019CC	
		IGG019TV	
		IGG019T2	
	0C	IGG019TV	
	10	IGG019FG	
	14	IGG019AL	
		IGG019DK	
		IGG019T2	
	18	IGG019AJ	
		IGG019BP	
		IGG019DJ	
		IGG019FJ	
003	01	IGG019CC	
	02	IGG019CE	
		IGG019CF	
		IGG019FK	

Abend Code	Return Code	Module	Message Number
	03	IGG019FA	
		IGG019FQ	
004	01	IGG0197N	
	02	IGG0197N	
	03	IGG0197N	
	04	IGG0197N	
	05	IGG0197M	
	06	IGG0197Q	
008		IGG019BS	
		IGG019DC	
013	10	IGG0191C	IEC141I
	14	IGG0191B	
	18	IGG0191B	
	1C	IGG0191B	
	20	IGG0191A	
	24	IGG0191A	
		IGG0199G	
	28	IGG0191A	
		IGG0199G	
	30	IGG0191F	
	34	IGG0191I	
		IGG0199G	
	40	IGG01910	
	48	IGG01911	
	4C	IGG0191A	
		IGG0196B	
		IGG0199G	
	50	IGG0196B	
	54	IGG0196A	
	5C	IGG01915	
		IGG01916	
		IGG0191I	
	60	IGG0191A	

Abend Code	Return Code	Module	Message Number
	70	IGG0199G	
	88	IGG0191B	
	8C	IGG0191L	
	90	IGG0197V	
	94	IGG0191A	
	98	IGG0191A	
	9C	IGG0197V	
	A0	IGG0191A	
	A4	IGG0199G	
	A8	IGG0199G	
	B0	IGG0191A	
	B4	IGG0191A	
	B8	IGG0197V	
	BC	IGG0199G	
	C0	IGG0199G	
	CC	IGG0196Q[1]	
	D0	IGG0191A	
	E4	IGG0196I	
112	01	IGCT0018	IEC908I
	02	IGCT0018	
	03	IGCT0018	
	04	IGCT0018	
	13	IGCT0018	
	14	IGCT0018	
115		IGG0210A	
118	01	IGCT0028	IEC912I
	02	IGCT0028	

Note

[1] Applies only to 3800 Printing Subsystem

Abend Code	Return Code	Module	Message Number
119	01	IGCT002E	IEC914I
	02	IGCT002E	
	03	IGCT002E	
	04	IGCT002E	
	12	IGCT002E	
	13	IGCT002E	
	14	IGCT002E	
144		IGC0006H	
145	01	IGCT0069	IEC916I
	02	IGCT0069	
	03	IGCT0069	
	04	IGCT0069	
	05	IGCT0069	
	06	IGCT0069	
	07	IGCT0069	
	08	IGC0006I	
		IGCT0069	
151		IGCT1081	IEC918I
169	01	IGCT010E	IEC919I
	02	IGCT010E	
	03	IGCT010E	
		IGC0010E[1]	
172		IGG019BX	
212	01	IGCT0018	IEC909I
	02	IGCT0018	
	03	IGCT0018	
215	01	IGCT0021	IEC910I
	02	IGCT0021	
	03	IGCT0021	
	04	IGCT0021	

Note

[1] Applies only to 3800 Printing Subsystem

Abend Code	Return Code	Module	Message Number
218	01	IGCT002D	IEC913I
	02	IGCT002D	
	03	IGCT002D	
219	01	IGCT002E	IEC915I
	02	IGCT002E	
	03	IGCT002E	
	04	IGCT002E	
244		IGCT006H	
245		IGCT0069	IEC917I
251	01	IGCT1081	IEC918I
	02	IGCT1081	
269		IGCT010E	IEC920I
315		IGCT0021	IEC911I
337	08	IGG019AV	IEC024I
344		IGCT006H	
351		IGCT1081	IEC918I
400		IGG019BX	
		IGG019BZ	
413	2C	IGG0191N	IEC145I
444		IGCT006H	
	01	IGCT1081	IEC918I
	02	IGCT1081	
	03	IGCT1081	
	04	IGCT1081	
544		IGCT006H	
644		IGCT006H	
744		IGCT006H	IEC907I
B13	04	IGG0196Q	IEC152I
		IGG0196R	
	08	IGG0196Q	
	0C	IGG0196Q	
		IGG0196R	
	10	IGG0196Q	

Abend Code	Return Code	Module	Message Number
	14	IGG0196Q	
	18	IGG0196Q	
	1C	IGG0196Q	
	20	IGG0196Q	
		IGG0196R	
	24	IGG0196Q	
		IGG0196R	
	28	IGG0196Q	
	2C	IGG0196Q	
	30	IGG0196Q	
	34	IGG0196Q	
B14	04	IGG0201B	IEC217I
		IGG0201Z	
	08	IGG0201B	
		IGG0201Z	
	0C	IGG0201B	
		IGG0201Z	
	10	IGG0201B	
		IGG0201Z	
	14	IGG0201B	
		IGG0201Z	
	18	IGG0201B	
		IGG0201Z	
C37		IGCT0055	IEC033I

SETPRT EXECUTOR RETURN/REASON CODES AND MESSAGES

The return codes produced by the SETPRT executors are set when an error is detected by one of the executors. The SETPRT work area contains the return code (at SPWRETCD) and the reason code (at SPWRSNCD) (see "Data Areas" on page 210 for a description of the SETPRT work area).

Executor IGC0008A detects the following errors:

Return Code	Reason Code	Message No.	Description
00000018			Either: • DCB not open • DCB invalid for a sequential data set • SETPRT parameter list invalid • Output device not UCS or 3800 printer
0000001C	00000000		Permanent I/O error in a previously initiated output operation.
0000001C	00000004		Possible lost data condition detected for a nonpage tracking printer.
00000048	00000004		Possible lost data condition detected for a page tracking printer.

Executor IGG08101 detects the following errors:

Return Code	Reason Code	Message No.	Description
00000004			Operator canceled job step—chain, train, or band not available.
00000008			Permanent I/O error during BLDL for UCS image.

Executor IGG08102 detects the following errors:

Return Code	Reason Code	Message No.	Description
0000000C		IEC126I	Permanent I/O error during UCS load.
00000010			Permanent I/O error during UCS verification display.
00000014			Operator canceled SETPRT during UCS verification display.
0000001C	00000004		Possible lost data condition detected for a nonpage tracking printer.

Executor IGG08103 detects the following errors:

Return Code	Reason Code	Message No.	Description
00000400			Operator canceled job step—FCB not available.
00000800			Permanent I/O error during BLDL for FCB image.
0000001C	00000004		Possible lost data condition detected for a nonpage tracking printer.
00000048	00000004		Possible lost data condition detected for a page tracking printer.

Executor IGG08104 detects the following errors:

Return Code	Reason Code	Message No.	Description
00000C00	00000000	IEC124I	Permanent I/O error during FCB load.
00000C00	00000004	IEC124I	Permanent I/O error during FCB load; LOAD check—probable FCB contents error.
00001000			Permanent I/O error during FCB verification display.
00001400			Operator canceled SETPRT during FCB verification display.
0000001C	00000004		Possible lost data condition detected for a nonpage tracking printer.
00000020			No storage available for FCB copy/convert area.
00000048	00000004		Possible lost data condition detected for a page tracking printer.

Executor IGG08105 detects the following errors:

Return Code	Reason Code	Message No.	Description
00000004			Operator canceled SETPRT—band not available.
00000008			Permanent I/O error during BLDL for UCS image table.

Return Code	Reason Code	Message No.	Description
0000000C			Permanent I/O error during band id display.
00000010			Permanent I/O error during UCSB read.
00000014			Operator canceled SETPRT during UCS verification display.
0000001C	00000004		Possible lost data condition detected for a nonpage tracking printer.
00000020			No storage available for read UCSB area.
00000048	00000004		Possible lost data condition detected for a page tracking printer.

Executor IGG08110 detects the following errors:

Return Code	Reason Code	Message No.	Description
00040000	0000nn04	IEC168I	Issued if any of the specified modules for character arrangement tables (where nn= 01 to 04) could not be found on the library.
00080000	0000nn04	IEC169I	An I/O error occurred while attempting to locate a character arrangement table (where nn= 01 to 04) in the library data set.
00000020		IEC174I	SYS1.IMAGELIB was not opened because storage space was not available.
00000024		IEC175I	SYS1.IMAGELIB cannot be opened.
00000038		IEC178I	I/O error occurred while trying to initialize the 3800 Printing Subsystem.

Executor ICC08111 detects the following errors:

Return Code	Reason Code	Message No.	Description
00040000	00000018	IEC168I	Issued if any of the specified modules for library character sets could not be found on the library.
00080000	00000018	IEC169I	An I/O error occurred while attempting to locate a library character set in the library data set.
000C0000	xx000018	IEC170I	An I/O error occurred while loading a library character set (xx indicates the opcode of the CCW when the error occurred).
000C0000	xx00001C	IEC170I	An I/O error occurred while loading WCGMs (xx indicates the opcode of the CCW when the error occurred).
00000030		IEC176I	There are more character set IDs requested in the character arrangement tables than the number of WCGMs installed on the printer.
00000044		IEC182I	A position in the translate table refers to a WCGM for which no character set has been specified.
0000004C	xx000018	IEC184I	A load check occurred while loading a library character set (xx indicates the opcode of the CCW when the error occurred).
0000004C	xx00001C	IEC184I	A load check occurred while loading WCGMs (xx indicates the opcode of the CCW when the error occurred).

Executor IGG08112 detects the following errors:

Return Code	Reason Code	Message No.	Description
00040000	00000008	IEC168I	The requested copy modification module could not be found on the library.
00040000	00mmnn10	IEC168I	Issued if any of the specified modules for graphic modification records could not be found on the library (nn is the index of the character arrangement table, and mm is the index of requested graphic character modification module within that table).

Return Code	Reason Code	Message No.	Description
00080000	00000008	IEC169I	An I/O error occurred while attempting to locate a copy modification module in the library data set.
00080000	00mmnn10	IEC169I	An I/O error occurred while attempting to locate a graphic character module (nn is the index of the character arrangement table, and mm is the index of requested graphic character modification module within that table.
000C0000	xx00nn04	IEC170I	An I/O error occurred while loading a translate table (where nn=01 to 04 and xx indicates the opcode of the CCW when the error occurred).
000C0000	xx000008	IEC170I	An I/O error occurred while loading a copy modification module (where xx indicates the opcode of the CCW when the error occurred).
000C0000	xxmmnn10	IEC170I	An I/O error occurred while loading a graphic character modification module (nn is the index of the character arrangement table, mm is the index of requested graphic character modification module within that table, and xx indicates opcode of the CCW when the error occurred).
00000020		IEC174I	SYS1.IMAGELIB was not opened because storage space was not available.
00000024		IEC175I	SYS1.IMAGELIB cannot be opened.
00000034		IEC177I	The requested copy modification module requires a translate table which was not loaded for this request.
0000004C	xx00nn04	IEC184I	A load check occurred while loading a translate table (where nn=01 to 04 and xx indicates the opcode of the CCW when the error occurred).
0000004C	xx000008	IEC184I	A load check occurred while loading a copy modification module (where xx indicates the opcode of the CCW when the error occurred).

Return Code	Reason Code	Message No.	Description
0000004C	xxmmnn10	IEC184I	A load check occurred while loading a graphic character modification module (nn is the index of the character arrangement table, mm is the index of requested graphic character modification module within that table, and xx indicates the opcode of the CCW when the error occurred).

Executor IGG08113 detects the following errors:

Return Code	Reason Code	Message No.	Description
00000400	00000020	IEC168I	The requested FCB image could not be located via the DCB exit list and was not in the library data set.
00000800	00000020	IEC169I	An I/O error occurred while attempting to locate the requested FCB image from the library data set.
00000C00	xx000020	IEC170I	An I/O error occurred while loading an FCB (where xx indicates the opcode of the CCW when the error occurred).
00001000	xx000020	IEC171I	An I/O error was encountered while printing a representative map of the requested FCB image.
00001400	00000020	IEC172I	SETPRT processing was terminated because the operator canceled the job after inspecting the representative FCB image as it was displayed on the 3800 Printing Subsystem.
00000020		IEC174I	SYS1.IMAGELIB was not opened because storage space was not available.
00000024		IEC175I	SYS1.IMAGELIB cannot be opened.
0000004C	xx000020	IEC184I	A load check occurred while loading a forms control buffer (xx indicates the opcode of the CCW when the error occurred).

Executor IGG08114 detects the following errors:

Return Code	Reason Code	Message No.	Description
00000028		IEC172I	The operator canceled SETPRT processing after receiving a request to load a forms overlay negative.
0000002C		IEC172I	The operator canceled SETPRT processing after receiving a request to rethread the 3800 Printing Subsystem.
0000003C		IEC179I	A request for threading to the burster-trimmer-stacker was issued but the feature is not installed on the 3800 Printing Subsystem being used.
00000040	xx000000	IEC180I	An I/O error occurred while trying to display a status code on the 3800 Printing Subsystem for a rethread or forms overlay request (xx indicates the CCW when the error occurred).
00000040	xx000000	IEC180I	An I/O error occurred while issuing a 'SENSE I/O' CCW to sense the present paper thread path from the 3800 Printing Subsystem (xx indicates the CCW when the error occurred).

Executor IGG08115 detects the following errors:

Return Code	Reason Code	Message No.	Description
000C0000	xx00000C	IEC170I	An I/O error was detected while loading the starting copy number in the 3800 Printing Subsystem.
000C0000	xx000014	IEC170I	An I/O error was detected while loading the total copy count and forms overlay image count into the 3800 Printing Subsystem.
00000040	xx000000	IEC180I	An I/O error occurred while resetting the 3800 Printing Subsystem to the first translate table.

Executor IGG08116 detects the following errors:

Return Code	Reason Code	Message No.	Description
00000048	xx000004	IEC183I	A system-restart-requested type of paper jam occurred on the 3800 Printing Subsystem (xx indicates the opcode of the CCW when the error occurred).
00000048	xx000008	IEC183I	Cancel key was pressed on the 3800 Printing Subsystem (xx indicates the opcode of the CCW when the condition occurred).

Executor IGG08117 detects the following errors:

Return Code	Reason Code	Message No.	Description
00000050	00000004	IEC181I	An invalid SETPRT request for a SYSOUT data segment was specified. A storage address was used for a copy modification module, character arrangement table, FCB, or user library DCB. Only 3800 Printing Subsystem load module IDs in SYS1.IMAGELIB are allowed for SYSOUT setup.
00000050	00000008	IEC185I	During SETPRT processing for a SYSOUT data segment, an error was detected while attempting to read a JFCB or JFCBE control block from SWA.
00000050	0000000C	IEC185I	During SETPRT processing for a SYSOUT data segment, an error was detected while invoking the CLOSE Subsystem Interface for the previous data segment.
00000050	00000010	IEC185I	During SETPRT processing for a SYSOUT data segment, an error was detected while invoking the OPEN Subsystem Interface for the new data segment being created.
00000050	00000014	IEC185I	During SETPRT processing for a SYSOUT data segment, an error was detected while the scheduler spool file allocation routine was segmenting the data set.
00000050	00000018	IEC185I	An ENQ macro, issued by SETPRT processing, failed.
00000050	0000001C	IEC185I	More than one DCB was open for the SYSOUT data set.

DEBUGGING EXCPVR CHANNEL PROGRAMS

The IOBs chained off the DCBIOBA pointer do not contain a channel program; they contain one or two CCWs and status information (in IOBNFLG1 and IOBEX). The CCWs and status information are used by the SIO/pagefix appendage to build a real-address channel program in the SAMB. The SAMB contains the channel program, the pagefix list, IDAW lists, the count field work area, and the status flags. Figure 38 shows the relationship between the DCB, the DEB, the ICQE, and the SAMB.

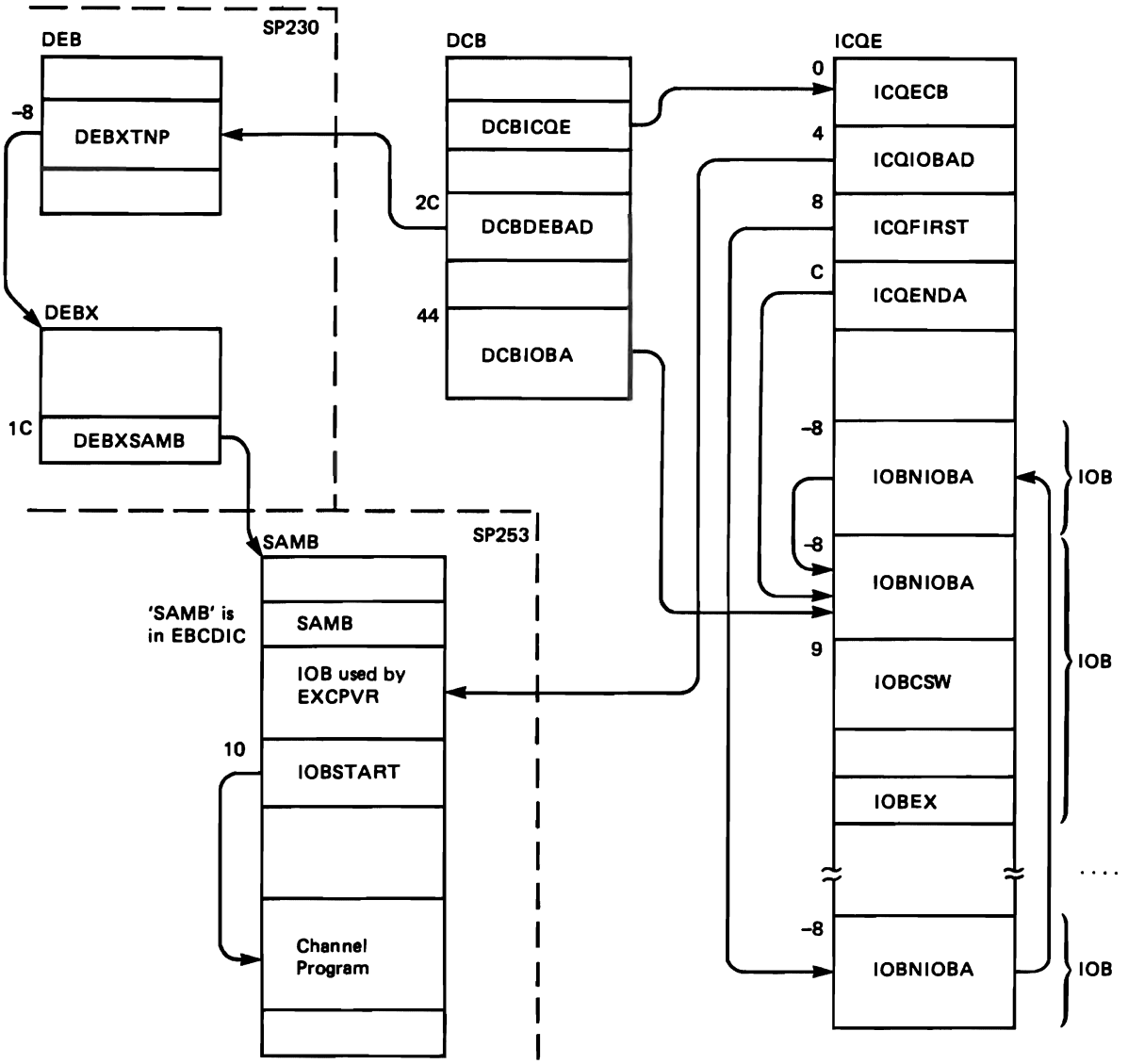


Figure 38. Control Blocks Used with EXCPVR Processing

APPENDIX A. BSAM/QSAM CHANNEL PROGRAMS

One real-address channel program is built in the SAMB. The channel program consists of a prolog (at SAMPROLG) and one or more channel program segments (at SAMCCW, which can contain a channel program of up to 27 CCWs). The channel program is chained from the prolog with a TIC (transfer in channel) command, and serves one of the IOBs on the active IOB queue (pointed to by the ICQE). There are five types of channel program segments:

Update-WRITE

Update-WRITE followed by refill-READ

Output (without the track overflow option)

Output (with the track overflow option)

Input

CHANNEL PROGRAM PROLOG SEGMENT

The prolog CCWs are partially built during OPEN processing. The IOBSTART field (in the SAMB's IOB) points to the first prolog CCW to be processed (see Note 3, below) when the channel program is executed.

For nonbuffered DASD devices, the following prefix is built.
Note: The prolog channel program segment issues a search ID EQ CCW to locate either:

- The desired record to be read or updated, or
- The record that immediately precedes the desired record to be read or updated (search previous logic).

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
PCCW1[1]	SET SECTOR	SAMSECT PCCW1+6[2]	CS	1 Sector[2]
PCCW2[1]	SEARCH ID EQ	IOBSEEK+3	C	5
PCCW3	TIC	PCCW2	—	—
PCCW4	TIC	SAMCCW[3]	—	—

For buffered DASD devices, the following prefix is built:

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
PCCW1	Unused	—	—	—
PCCW2	Unused	—	—	—
PCCW3	LOCATE RECORD	SAMLRPRM	C	16
PCCW4	TIC	SAMCCW	—	—

IOBSTART points to PCCW3.

For buffered DASD, all occurrences of (set sector), search id EQ, TIC*-8 will be replaced by a located record.

FLAGS

C = Command chain

S = SLI

Notes:

1 IOBSTART points to PCCW1 if the device includes the rotational position sensing feature. Otherwise, IOBSTART points to PCCW2.

2 When the request is update-WRITE, the sector value is the first byte of PCCW1's count field. Otherwise, it is the SAMSECT field.

3 PCCW4 might branch (with a TIC) to a CCW other than the first CCW in SAMCCW:

—When a re-EXCPVR exit from the channel end/abnormal end appendage occurs, or

—When a return from the first entry to the abnormal end appendage occurs.

UPDATE-WRITE CHANNEL PROGRAM SEGMENT

The update-WRITE channel program segment is built to serve an update-WRITE-only request (that is, SAMBSWR in SAMFLAG1 is on). The channel program segment is built by the CCWBLDUP routine (in module IGG019BX). Only one BSAM update-WRITE channel program segment is built in SAMCCW at a time.

If the write validity option is not specified, the update-WRITE channel program is:

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	WRITE DATA	Buffer[1]	CI[1]	—
CCW2	NOP	—	S	1

FLAGS

C = Chain command

S = SLI

I = Indirect addressing

Note:

1 If the data set resides on a VIO device or if the user's buffer is in a V=R address space, the address field contains the virtual-storage buffer address and no IDA flag is set. Otherwise, the address field contains the address of an IDA list and the IDA flag is set.

If the write validity check option is specified and the data set being accessed is not a track overflow data set, the update-WRITE channel program is:

WITHOUT ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	WRITE DATA	Buffer[1]	CI[1]	—
CCW2	SEARCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	CCW2	—	—
CCW4	READ KEY DATA	0	CSK	X'7FFF'
CCW5	NOP	—	S	—

WITH ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	WRITE DATA	Buffer[1]	CI[1]	—
CCW2	READ SECTOR	PCCW1+6[2]	CS	1
CCW3	SET SECTOR[3]	PCCW1+6[2]	CS	1
CCW4	SEARCH ID EQ[3]	IOBSEEK+3	C	5
CCW5	TIC[3]	CCW4	—	—
CCW6	READ KEY DATA	0	CSK	X'7FFF'
CCW7	NOP	—	S	—

FLAGS

C = Chain command

S = SLI

K = Skip

I = Indirect addressing

Notes:

[1] If the data set resides on a VIO device or if the user's buffer is in a V=R address space, the address field contains the virtual-storage buffer address and no IDA flag is set. Otherwise, the address field contains the address of an IDA list and the IDA flag is set.

[2] PCCW1 is the prolog channel program segment's first CCW, a set-sector command.

[3] For buffered DASD, CCWs 3, 4, and 5 will be replaced by the following:

| CCW3 | LOCATE RECORD | SAMLRPRM+16 | C | 16 |

Some of the data records of a track overflow data set are written using write special CKD CCWs. Consequently, the direct-access device automatically switches to the next track and cylinder when the end of each track overflow record segment (except the last) is detected during a write data or read data CCW. If the WRITE validity option is specified and a track overflow data set is being accessed, the update-WRITE channel program is:

WITHOUT ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	WRITE DATA	Buffer[1]	CI[1]	—
CCW2	SEEK HEAD[2]	IOBSEEK+1	C	6
	NOP[2]	—	CS	1
CCW3	SEARCH ID EQ	IOBSEEK+3	C	5
CCW4	TIC	CCW3	—	—
CCW5	READ KEY DATA	0	CSK	X'FFF'
CCW6	NOP	—	S	1

WITH ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	WRITE DATA	Buffer[1]	CI[1]	—
CCW2	SEEK HEAD[2]	IOBSEEK+1	C	6
	NOP[2]	—	CS	1
CCW3	SET SECTOR[3]	PCCW1+6[4]	CS	1
CCW4	SEARCH ID EQ[3]	IOBSEEK+3	C	5
CCW5	TIC[3]	CCW4	—	—
CCW6	READ KEY DATA	0	CSK	X'7FFF'
CCW7	NOP	—	S	1

FLAGS

C = Chain command

S = SLI

K = Skip

I = Indirect addressing

Notes:

1 If the data set resides on a VIO device or if the user's buffer is in a V=R address space, the address field contains the virtual-storage buffer address and no IDA flag is set. Otherwise, the address field contains the address of an IDA list and the IDA flag is set.

2 The seek head command is part of the channel program if the DEB extent file mask permits head seeks. Otherwise, the seek head command is replaced with a NOP command.

3 For buffered DASD, CCWs 3, 4, and 5 will be replaced by the following:

| CCW3 | LOCATE RECORD | SAMLRPRM+16 | C | 16 |

4 PCCW1 is the prolog channel program segment's first CCW, a set sector command.

UPDATE-WRITE FOLLOWED BY REFILL-READ CHANNEL PROGRAM SEGMENT

The update-WRITE followed by refill-READ channel program segment is built to serve a request that updates a record and then reads a subsequent record (not necessarily the next sequential record) into the buffer. This type of request is indicated with the SAMPUTX bit in SAMFLAG1 set on. The segment is built by the CCWBLDUP and CCWBLDIP routines (in module IGG019BX).

Only one update-WRITE followed by a refill-READ channel program segment is built in SAMCCW at a time.

If the write validity check option is not specified, the update-write followed by refill-read channel program segment is:

WITHOUT ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	WRITE DATA	Buffer[1]	CI[1]	—
CCW2	SEEK HEAD[2]	SAMSEEK+1	C	6
	NOP[2]	—	CS	1
CCW3	SEARCH ID EQ	SAMSEEK+3	C	5
CCW4	TIC	CCW3	—	—
CCW5	READ KEY DATA[3]	0	CSK	X'7FFF'
CCW6	M/T READ COUNT[4]	SAMCNTS[5]	C	8
CCW7	READ DATA	Buffer[1]	CI[1]S[6]	—
CCW8	M/T READ COUNT	SAMCNTS[5]	C	8
CCW9	NOP	—	S	1

WITH ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	WRITE DATA	Buffer[1]	CI[1]	—
CCW2	SEEK HEAD[2]	SAMSEEK+1	C	6
	NOP[2]	—	CS	1
CCW3	SET SECTOR[7]	SAMSECT	CS	1
CCW4	SEARCH ID EQ[7]	SAMSEEK+3	C	5
CCW5	TIC[7]	CCW4	—	—
CCW6	READ KEY DATA[3]	0	CSK	X'7FFF'
CCW7	M/T READ COUNT[4]	SAMCNTS[5]	C	8
CCW8	READ DATA	Buffer[1]	CI[1]S[6]	—
CCW9	M/T READ COUNT	SAMCNTS	C	8
CCW10	READ SECTOR	SAMSECT	S	1

FLAGS

C = Chain command

S = SLI

K = Skip

I = Indirect addressing

Notes:

1. If the data set resides on a VIO device or if the user's buffer is in a V=R address space, the address field contains the virtual-storage buffer address and no IDA flag is set. Otherwise, the address field contains the address of an IDA list and the IDA flag is set.

2. The seek head command is part of the channel program if the DEB extent file mask permits head seeks. Otherwise, the seek head command is replaced with a NOP command.

3. The read key data CCW is part of the channel program only when search previous logic is required for a track-overflow data set. Otherwise, the CCW is omitted.

4. The M/T read count. CCW is part of the channel program only when search previous logic is required for the data set (either a track-overflow data set or not). Otherwise, the CCW is omitted.

[5] The count field used in this CCW is one of the 14 eight-byte fields in SAMCNTS (in the SAMB control block), and is different from other SAMCNTS count fields used by other CCWs in the channel program.

[6] The SLI flag is set to 1 if the record format is variable length (that is, RECFM=V). The SLI flag is also set to 1 if:

- The record format is not fixed-length records written as standard blocks (RECFM=FS or RECFM=FBS), and
- The data set is not a track-overflow data set (RECFM=T).

[7] For buffered DASD, CCWs 3, 4, and 5 will be replaced by the following:

| CCW3 | LOCATE RECORD | SAMLRPRM+16 | C | 16 |

If the WRITE validity check option is specified and the data set being accessed is not a track overflow data set, the update-WRITE channel program segment is:

WITHOUT ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	WRITE DATA	Buffer[1]	CI[1].	—
CCW2	SEARCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	CCW4	—	—
CCW4	READ KEY DATA	0	CSK	X'7FFF'
CCW5	SEEK HEAD[2]	SAMSEEK+1	C	6
	NOP[2]	—	CS	1
CCW6	SEARCH ID EQ	SAMSEEK+3	C	5
CCW7	TIC	CCW6	—	—
CCW8	M/T READ COUNT[3]	SAMCNTS[4]	C	8
CCW9	READ DATA	Buffer[1]	CI[1]S[5]	—
CCW10	M/T READ COUNT	SAMCNTS[4]	C	8
CCW11	NOP	—	—	1

WITH ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	WRITE DATA	Buffer[1]	CI[1]	—
CCW2	READ SECTOR	PCCW1+6[6]	CS	1
CCW3	SET SECTOR	PCCW1+6[6]	CS	1
CCW4	SEARCH ID EQ	IOBSEEK+3	C	5
CCW5	TIC	CCW4	—	—
CCW6	READ KEY DATA	0	CSK	X'7FFF'
CCW7	SEEK HEAD[2]	SAMSEEK+1	C	6
	NOP[2]	—	CS	1
CCW8	SET SECTOR[7]	SAMSECT	CS	1
CCW9	SEARCH ID EQ[7]	SAMSEEK+3	C	5
CCW10	TIC[7]	CCW9	—	—
CCW11	M/T READ COUNT[3]	SAMCNTS[4]	C	8
CCW12	READ DATA	Buffer[1]	CI[1]S[5]	—
CCW13	M/T READ COUNT	SAMCNTS[4]	C	8
CCW14	READ SECTOR	SAMSECT	S	1

FLAGS

C = Chain command

S = SLI

K = Skip

I = Indirect addressing

Notes:

1. If the data set resides on a VIO device or if the user's buffer is in a V=R address space, the address field contains the virtual-storage buffer address and no IDA flag is set. Otherwise, the address field contains the address of an IDA list and the IDA flag is set.

2. The seek head command is part of the channel program if the DEB extent file mask permits head seeks. Otherwise, the seek head command is replaced with a NOP command.

3. The M/T read count. CCW is part of the channel program only when search previous logic is required. Otherwise, the CCW is omitted.

4. The count field used in this CCW is one of the 14 eight-byte fields in SAMCNTS (in the SAMB control block), and is different from other SAMCNTS count fields used by other CCWs in the channel program.

5. The SLI flag is set to 1 if the record format is variable-length (that is, RECFM=V). The SLI flag is also set to 1 if the record format is not fixed-length records written as standard blocks (RECFM=FS or RECFM=FBS).

6. PCCW1 is the prolog channel program segment's first CCW, a set-sector command.

7. For buffered DASD, CCWs 8, 9, and 10 will be replaced by the following:

   | CCW3 | LOCATE RECORD | SAMLRPRM+16 | C | 16 |

Some of the data records of a track-overflow data set are written using write special CKD CCWs. Consequently, the direct-access device automatically switches to the next track and cylinder when the end of each track-overflow record segment (except the last) is detected during a write data or read data CCW. If the WRITE validity check option is specified and a track-overflow data set is being accessed, the update-WRITE with refill READ channel program segment is as follows:

WITHOUT ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	WRITE DATA	Buffer[1]	CI[1]	—
CCW2	SEEK HEAD[2]	IOBSEEK+1	C	5
	NOP[2]	—	CS	1
CCW3	SEARCH ID EQ	IOBSEEK+3	C	5
CCW4	TIC	CCW3	—	—
CCW5	READ KEY DATA	0	CSK	X'7FFF'
CCW6	SEEK HEAD[2]	SAMSEEK+1	C	6
	NOP[2]	—	CS	1
CCW7	SEARCH ID EQ	SAMSEEK+3	C	5
CCW8	TIC	CCW7	—	—
CCW9	READ KEY DATA[3]	0	CSK	X'7FFF'
CCW10	M/T READ COUNT[3]	SAMCNTS[4]	C	8
CCW11	READ DATA	Buffer[1]	CI[1]S[5]	—
CCW12	M/T READ COUNT	SAMCNTS[4]	C	8
CCW13	NOP	—	—	1

WITH ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	WRITE DATA	Buffer[1]	CI[1]	—
CCW2	SEEK HEAD[2]	IOBSEEK+1	C	6
	NOP[2]	—	CS	1
CCW3	SET SECTOR	PCCW+6[6]	CS	1
CCW4	SEARCH ID EQ	IOBSEEK+3	C	5
CCW5	TIC	CCW4	—	—
CCW6	READ KEY DATA	0	CSK	X'7FFF'
CCW7	SEEK HEAD[2]	SAMSEEK+1	C	6
	NOP[2]	—	CS	1
CCW8	SET SECTOR	SAMSECT	CS	1
CCW9	SEARCH ID EQ	SAMSEEK+3	C	5
CCW10	TIC	CCW9	—	—
CCW11	READ KEY DATA[3]	0	CSK	X'7FFF'
CCW12	M/T READ COUNT[3]	SAMCNTS[4]	C	8
CCW13	READ DATA	Buffer[1]	CI[1]S[5]	—
CCW14	M/T READ COUNT	SAMCNTS[4]	C	8
CCW15	READ SECTOR	SAMSECT	S	1

FLAGS

C = Chain command

S = SLI

K = Skip

I = Indirect addressing

Notes:

1. If the data set resides on a VIO device or if the user's buffer is in a V=R address space, the address field contains the virtual-storage buffer address and no IDA flag is set. Otherwise, the address field contains the address of an IDA list and the IDA flag is set.

2. The seek head command is part of the channel program if the DEB extent file mask permits head seeks. Otherwise, the seek head command is replaced with a NOP command.

3. The read key data and M/T read count. CCWs are part of the channel program only when search previous logic is required. Otherwise, these two CCWs are omitted.

4. The count field used in this CCW is one of the 14 eight-byte fields in SAMCNTS (in the SAMB control block), and is different from other SAMCNTS count fields used by other CCWs in the channel program.

5. The SLI flag is set to 1 if the record format is variable-length (that is, RECFM=V).

6. PCCW1 is the prolog channel program segment's first CCW, a set sector command.

OUTPUT CHANNEL PROGRAM SEGMENT (TO WRITE OUTPUT RECORDS THAT ARE NOT TRACK OVERFLOW RECORDS)

The output channel program segment is built to serve a request that writes an output record to a data set that is not a track-overflow data set. Output channel program segments can be chained together to write successive records to the data set. The output channel program segment is built by the CCWBLDOP routine (in module IGG019BX).

If the write validity check option is not specified, the output channel program segment is:

WITHOUT ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	WRITE CKD	SAMCNTS[1]	CD	8
CCW2	Buffer[2]	CI[2]	—	
CCW3	NOP	—	S	1

WITH ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	WRITE CKD	SAMCNTS[1]	CD	8
CCW2		Buffer[2]	CI[2]	—
CCW3	READ SECTOR	SAMSECT	S	1

FLAGS

C = Chain command

S = SLI

D = Chain data

I = Indirect addressing

Note:

[1] The count field used in this CCW is one of the 14 eight-byte fields in SAMCNTS (in the SAMB control block), and is different from other SAMCNTS count fields used by other CCWs in the channel program (unless otherwise noted).

[2] If the data set resides on a VIO device or if the user's buffer is in a V=R address space, the address field contains the virtual-storage buffer address and no IDA flag is set. Otherwise, the address field contains the address of an IDA list and the IDA flag is set.

If the WRITE validity check option is specified, the output channel program segment is:

WITHOUT ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	WRITE CKD	SAMCNTS[1]	CD	8
CCW2	Buffer[2]	CI[2]	—	
CCW3	SEARCH ID EQ	SAMCNTS[3]	C	5
CCW4	TIC	CCW3	—	—
CCW5	READ KEY DATA	0	CSK	X'7FFF'
CCW6	NOP	—	S	1

WITH ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	WRITE CKD	SAMCNTS[1]	CD	8
CCW2	Buffer[2]	CI[2]	—	
CCW3	READ SECTOR	SAMSECT	CS	1
CCW4	SET SECTOR	SAMSECT	CS	1
CCW5	SEARCH ID EQ	SAMCNTS[3]	C	5
CCW6	TIC	CCW5	—	—
CCW7	READ KEY DATA	0	CSK	X'7FFF'
CCW8	READ SECTOR	SAMSECT	S	1

WITH BUFFERRED DASD

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	WRITE CKD	SAMCNTS[1]	CD	8
•	• •	BUFFER	CI	—
•	• •	•	•	
•	• •	•	•	
CCWn	[MT] WRITE CKD	SAMCNTS[1]	CD	8
		BUFFER	CI	—
CCWn	READ SECTOR	SAMSECT	S	1
CCWn	LOCATE RECORD	SAMLRPRM+16	C	16
CCWn	[MT] READ KEY DATA	0	CSK	X'7FFF'
•	• •	•	•	
•	• •	•	•	
•	• •	•	•	
CCWn	[MT] READ KEY DATA	0	SK	X'7FFF'

FLAGS

C = Chain command

S = SLI

K = Skip

D = Chain data

I = Indirect addressing

Notes:

[1] The count field used in this CCW is one of the 14 eight-byte fields in SAMCNTS (in the SAMB control block), and is different from other SAMCNTS count fields used by other CCWs in the channel program (unless otherwise noted).

[2] If the data set resides on a VIO device or if the user's buffer is in a V=R address space, the address field contains the virtual-storage buffer address and no IDA flag is set. Otherwise, the address field contains the address of an IDA list and the IDA flag is set.

[3] The SAMCNTS field is the same one used by CCW1, the first CCW in the channel program segment.

The preceding output channel program segments can be chained together to write successive records all on one track. The last CCW (that is, the NOP or read sector CCW) of each channel program segment except the last is omitted:

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT	
CCW1	WRITE CKD	SAMCNTS	CD	8	Write first record
CCW2		Buffer[1]	CI[1]	—	
CCW3	WRITE CKD	SAMCNTS+8	CD	8	Write second record
CCW4		Buffer[1]	CI[1]	—	
. . .					. . .
CCW9	WRITE CKD	SAMCNTS+32	CD	8	Write last record
CCW10		Buffer[1]	CI[1]	—	
CCW11	NOP[2]	—	S	1	
	READ SECTOR[2]	SAMSECT	S	1	

FLAGS

C = Chain command

S = SLI

D = Chain data

I = Indirect addressing

Notes:

[1] If the data set resides on a VIO device or if the user's buffer is in a V=R address space, the address field contains the virtual-storage buffer address and no IDA flag is set. Otherwise, the address field contains the address of an IDA list and the IDA flag is set.

[2] The read sector CCW is present if the device includes the rotational position sensing feature. Otherwise, the NOP CCW is present.

If the second (or subsequent) record is to be written as record R1 on the next track, a multitrack search CCW orients the channel program:

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT	
CCW1	WRITE CKD	SAMCNTS	CD	8	Write first record
CCW2		Buffer[1]	CI[1]	—	
CCW3[4]	M/T SRCH ID EQ	SAMCNTS+8[2]	C	5	Locate record 0 of the next track
CCW4[4]	TIC	CCW3	—	—	
CCW5	WRITE CKD	SAMCNTS+16	CD	8	Write second record
CCW6		Buffer[1]	CI[1]	—	
. . .					. . .
CCW11	WRITE CKD	SAMCNTS+40	CD	8	Write last record
CCW12		Buffer[1]	CI[1]	—	
CCW13	NOP[3]	—	S	1	
	READ SECTOR[3]	SAMSECT	S	1	

FLAGS

C = Chain command

S = SLI

D = Chain data

I = Indirect addressing

Notes:

1. If the data set resides on a VIO device or if the user's buffer is in a V=R address space, the address field contains the virtual-storage buffer address and no IDA flag is set. Otherwise, the address field contains the address of an IDA list and the IDA flag is set.

2. The field in SAMCNTS contains the CCHHR of the record to be written (record R1) with R set to 0, causing a search for record zero on the next track.

3. The read sector CCW is present if the device includes the rotational position sensing feature. Otherwise, the NOP CCW is present.

4. CCW3 and CCW4 are present only when writing record R1 on a track when the record is not the first record written in the channel program.

OUTPUT CHANNEL PROGRAM SEGMENT (TO WRITE TRACK OVERFLOW RECORDS)

The output channel program segment is built to serve a request that writes a record to a track-overflow data set. If the track-overflow record is segmented, output channel program segments are chained together to write all segments of the record.

Module IGG019T2 determines whether enough room remains on the track to write the record. If not, IGG019T2 separates the track-overflow record into segments and determines the length of each segment. IGG019T2 also determines whether the track-overflow record can be written in the extent's remaining space.

Only one output record is written at a time, even though many output channel program segments are chained together to write all track overflow segments of the record. The track-overflow record output channel program is built by the CCWBLDOT routine (in module IGG019BX).

If the record does not overflow to another track (that is, it fits entirely on the remaining space on the track) and the WRITE validity check option is not specified, the output channel program segment that writes the record is the same as described previously for writing output records that are not track-overflow records.

If the record does not overflow and the WRITE validity option is specified, the output channel program segment that writes the record is:

WITHOUT ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	WRITE CKD	SAMCNTS	CD	8
CCW2		Buffer[1]	CI[1]	—
CCW3	SEEK HEAD[2]	SAMCNTS[3]	C	6
	NOP[2]	—	C	1
CCW4	SEARCH ID EQ	SAMCNTS[3]	C	5
CCW5	TIC	CCW4	—	—
CCW6	READ KEY DATA	0	CSK	X'7FFF'
CCW7	NOP	—	S	1

WITH ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	WRITE CKD	SAMCNTS	CD	8
CCW2		Buffer[1]	CI[1]	—
CCW3	SEEK HEAD[2]	SAMCNTS[3]	C	6
	NOP[2]	—	C	1
CCW4	SET SECTOR	CCW4+6	CS	Sector[4]|1
CCW5	SEARCH ID EQ	SAMCNTS[3]	C	5
CCW6	TIC	CCW5	—	—
CCW7	READ KEY DATA	0	CSK	X'7FFF'
CCW8	READ SECTOR	SAMSECT	S	1

Notes and Flags explanation follow:

FLAGS

C = Chain command

S = SLI

K = Skip

D = Chain data

I = Indirect addressing

Notes:

1 If the data set resides on a VIO device or if the user's buffer is in a V=R address space, the address field contains the virtual-storage buffer address and no IDA flag is set. Otherwise, the address field contains the address of an IDA list and the IDA flag is set.

2 The seek head command is part of the channel program if the DEB extent file mask permits head seeks (although it has no effect in this case), the record doesn't overflow to another track. Otherwise, the seek head command is replaced with a NOP command.

3 The SAMCNTS field is the same one used by the first segment's Write CCW (CCW1).

4 The sector value is placed in the first byte of the CCW's count field. The sector value is the SAMSECT value if the first overflow record is written by this channel program. Otherwise, the sector value is 0.

If the track-overflow record consists of more than one segment (that is, it overflows to another track), one or more write special CKD CCWs are used to write the record. When the end of a record written with a write special CKD CCW is read, the direct-access device automatically switches to the next track or cylinder so that the entire record is read with one READ CCW.

The first write special CKD CCW writes the record's count field and data on the remainder of the track: the track contains the record's first overflow segment. An M/T (multitrack) search ID equal CCW orients the channel program to the start of the next track or cylinder (that is, immediately after the count field of record 0). Subsequent write special CKD CCWs write track-overflow record segments that occupy the entire track. The record's last segment is written with a write CKD CCW, and occupies the first part of the track.

If the WRITE validity option has been specified, a seek head CCW, a set sector CCW, and a search ID equal CCW reposition the direct-access device to read the record's first overflow segment. Because the overflow segments are written using write

special CKD CCWs, the WRITE validity's read key data CCW reads all segments of the record: The direct-access device automatically switches tracks and cylinders when the end of each overflow segment is read. Note that the record's last segment is not an overflow segment and does not cause track switching.

Before this channel program is built and issued, IGG019T2 has determined that the entire record can fit in the space available (that is, it can fit in the extent). When the track-overflow record cannot fit in the extent, IGG019T2 directs the CCWBLDOT routine to build a channel program identical to the one shown below, except that all write special CKD and write CKD CCWs are replaced with erase CCWs. IGG019T2 waits until the ERASE channel program completes, then locates the data set's next extent. It next recalculates the length of each track-overflow segment to be written and directs the CCWBLDOT routine to build a channel program that writes the complete record.

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT	
CCW1	WRITE SPCL CKD	SAMCNTS	CD	8	Write record's first overflow segment.
CCW2		Buffer[1]	CI[1]	—	
CCW3	M/T SRCH ID EQ	SAMCNTS	C	5	Switch to next track.
CCW4	TIC	CCW3	—	—	
CCW5	WRITE SPCL CKD	SAMCNTS	CD	8	Write record's second overflow segment.
CCW6		Buffer[1]	CI[1]	—	
. . .					. . .
CCW15	M/T SRCH ID EQ	SAMCNTS+32	C	5	Switch to next track.
CCW16	TIC	CCW15	—	—	
CCW17	WRITE CKD	SAMCNTS+32	CD	8	Write record's last segment.
CCW18		Buffer[1]	CI[1]	—	
CCW19[2]	SEEK HEAD	SAMCNTS[3]	C	6	Reposition to record's first segment.
CCW20[2]	SET SECTOR[4]	CCW4+6	CS	Sector[5]|[1]	
CCW21[2]	SEARCH ID EQ	SAMCNTS[3]	C	5	
CCW22[2]	TIC	CCW5	—	—	
CCW23[2]	READ KEY DATA	0	CSK	X'7FFF'	Write validity check.
CCW24	READ SECTOR[4]	SAMSECT	S	1	
	NOP[2]	—	S	1	

FLAGS

C = Chain command

S = SLI

D = Chain data

I = Indirect addressing

Notes:

1 If the data set resides on a VIO device or if the user's buffer is in a V=R address space, the address field contains the virtual-storage buffer address and no IDA flag is set. Otherwise, the address field contains the address of an IDA list and the IDA flag is set.

2 CCW19 through CCW23 are included in the channel program when the write validity check option is specified. Otherwise, CCW19 through CCW23 are omitted.

[3] The SAMCNTS field is the same one used by the first write CCW (CCW1).

[4] The read sector CCW is present if the device includes the rotational position sensing feature. Otherwise, CCW20 is omitted and is not replaced with a NOP CCW, and CCW24 is replaced with a NOP CCW.

[5] The sector value is placed in the first byte of the CCW's count field. The sector value is the SAMSECT value if the first overflow record is written by this channel program. Otherwise, the sector value is 0.

INPUT CHANNEL PROGRAM SEGMENT

The input channel program segment is built to serve a request to read all segments of a record. The channel program segment reads records of data sets opened for UPDAT, INPUT, INOUT, and OUTIN processing. Input channel program segments can be chained together to read more than 1 record. The input channel program segment is built by the CCWBLDIP routine (in module IGG019BX).
Note: The prolog channel program segment issues a search ID EQ CCW to orient to either:

- The record to be read, or
- The record immediately preceding the record to be read (search previous logic).

When the prolog's search CCW locates the record to be read, the input channel program segment is:

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	READ DATA[1]	Buffer[2]	CI[2]S[3]	—
CCW2	M/T READ COUNT[4]	SAMCNTS[5]	C	8
CCW3	READ SECTOR[6]	SAMSECT	S	1
	NOP[6]	—	S	1

FLAGS

C = Chain command

S = SLI

I = Indirect addressing

Notes:

[1] If the record has a key area and the key is to be read (BSAM only), the read key data CCW is used instead of the read data CCW.

[2] If the data set resides on a VIO device or if the user's buffer is in a V=R address space, the address field contains the virtual-storage buffer address and no IDA flag is set. Otherwise, the address field contains the address of an IDA list and the IDA flag is set.

[3] The SLI flag is set to 1 if the record format is variable-length (that is, RECFM=V). The SLI flag is also set to 1 if:

- The record format is not fixed-length records written as standard blocks (RECFM=FS or RECFM=FBS), and
- The data set is not a track-overflow data set (RECFM=T).

[4] CCW2 and CCW3 read the count field and the sector value of the next sequential record. (The next record might be on the same track, or it might be record 1 on the next track, next cylinder, or next extent.) When the user's program requests the next input record, the prolog's search CCW can locate it—minimizing the number of times search-previous logic is required.

[5] The count field used in this CCW is one of the 14 eight-byte fields in SAMCNTS (in the SAMB control block), and is different from other SAMCNTS count fields used by other CCWs in the channel program.

[6] The read sector CCW is present if the device includes the rotational position sensing feature. Otherwise, the NOP CCW is present.

When the prolog's search CCW locates the record immediately preceding the record to be read and the data set being accessed is not a track-overflow data set, the input channel program segment is:

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	M/T READ COUNT	SAMCNTS[1]	C	8
CCW2	READ DATA[2]	Buffer[3]	CI[3]S[4]	—
CCW3	M/T READ COUNT[5]	SAMCNTS[1]	C	8
CCW4	READ SECTOR[6]	SAMSECT	S	1
	NOP[6]	—	S	1

FLAGS

C = Chain command

S = SLI

I = Indirect addressing

Notes:

[1] The count field used in this CCW if one of the 14 eight-byte fields in SAMCNTS (in the SAMB control block), and is different from other SAMCNTS count fields used by other CCWs in the channel program.

[2] If the record has a key area and the key is to be read (BSAM only), the read key data CCW is used instead of the read data CCW.

[3] If the data set resides on a VIO device or if the user's buffer is in a V=R address space, the address field contains the virtual-storage buffer address and no IDA flag is set. Otherwise, the address field contains the address of an IDA list and the IDA flag is set.

[4] The SLI flag is set to 1 if the record format is variable-length (that is, RECFM=V). The SLI flag is also set to 1 if the record format is not fixed-length records written as standard blocks (RECFM=FS).

[5] CCW3 reads the count field of the next sequential record. (The next record might be on the same track, or it might be record one on the next track, next cylinder, or next extent.) When the user's program requests the next input record, the prolog's search CCW can locate it—minimizing the number of times search-previous logic is required.

[6] The read sector CCW is present if the device includes the rotational position sensing feature. Otherwise, the NOP CCW is present.

Some of the data records of a track-overflow data set are written using write special CKD CCWs. Consequently, the direct-access device automatically switches to the next track and cylinder when the end of each track-overflow record segment is read with a read data or read key data CCW. When the prolog's search CCW locates the record immediately preceding the required record (that is, search-previous logic is used) and a track-overflow data set is being read, the input channel program segment is:

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	READ KEY DATA	0	CSK	X'7FFF'
CCW2	M/T READ COUNT	SAMCNTS[1]	C	8
CCW3	READ DATA[2]	Buffer[3]	CI[3]S[4]	—
CCW4	M/T READ COUNT[5]	SAMCNTS[1]	C	8
CCW5	READ SECTOR[6]	SAMSECT	S	1
	NOP[6]	—	S	1

FLAGS

C = Chain command

S = SLI

K = Skip

I = Indirect addressing

Notes:

1 The count field used in this CCW is one of the 14 eight-byte fields in SAMCNTS (in the SAMB control block), and is different from other SAMCNTS count fields used by other CCWs in the channel program.

2 If the record has a key area and the key is to be read (BSAM only), the read key data CCW is used instead of the read data CCW.

3 If the data set resides on a VIO device or if the user's buffer is in a V=R address space, the address field contains the virtual-storage buffer address and no IDA flag is set. Otherwise, the address field contains the address of an IDA list and the IDA flag is set.

4 The SLI flag is set to 1 if the record format is variable length (that is, RECFM=V).

5 CCW4 reads the count field of the next sequential record. (The next record might be on the same track, or it might be record one on the next track, next cylinder, or next extent.) When the user's program requests the next input record, the prolog's search CCW can locate it—minimizing the number of times search-previous logic is required.

6 The read sector CCW is present if the device includes the rotational position sensing feature. Otherwise, the NOP CCW is present.

An example of a channel program that reads five records when search-previous logic is required to locate the first record is:

DATA SET INCLUDES THE TRACK-OVERFLOW OPTION

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT	
CCW1	READ KEY DATA	0	CSK	X'7FFF'	Search previous track to locate first record.
CCW2	M/T READ COUNT	SAMCNTS	C	8	
CCW3	READ DATA[1]	Buffer[2]	CI[2]S[3]	—	Read first record.
CCW4	M/T READ COUNT	SAMCNTS+8	C	8	
CCW5	READ DATA[1]	Buffer[2]	CI[2]S[3]	—	Read second record.
CCW6	M/T READ COUNT	SAMCNTS+16	C	*	
. . .					. . .
CCW11	READ DATA[1]	Buffer[2]	CI[2]S[3]	—	Read last record.
CCW12	M/T READ COUNT	SAMCNTS+40	C	8	
CCW13	READ SECTOR[4]	SAMSECT	S	1	
	NOP[4]	—	S	1	

DATA SET DOESN'T INCLUDE THE TRACK-OVERFLOW OPTION

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT	
CCW1	M/T READ COUNT	SAMCNTS	C	8	Locate and read first record.
CCW2	READ DATA[1]	Buffer[2]	CI[2]S[3]	—	
CCW3	M/T READ COUNT	SAMCNTS+8	C	8	
CCW4	READ DATA[1]	Buffer[2]	CI[2]S[3]	—	Read second record.
CCW5	M/T READ COUNT	SAMCNTS+16	C	8	
. . .					. . .
CCW10	READ DATA[1]	Buffer[2]	CI[2]S[3]	—	Read last record.
CCW11	M/T READ COUNT[5]	SAMCNTS+40	C	8	
CCW13	READ SECTOR[4]	SAMSECT	S	1	
	NOP[4]	—	S	1	

FLAGS

C = Chain command

S = SLI

K = Skip

I = Indirect addressing

Notes:

1 If the record has a key area and the key is to be read (BSAM only), the read key data CCW is used instead of the read data CCW.

2 If the data set resides on a VIO device or if the user's buffer is in a V=R address space, the address field contains the virtual-storage buffer address and no IDA flag is set. Otherwise, the address field contains the address of an IDA list and the IDA flag is set.

3 The SLI flag is set to 1 if the record format is variable length (that is, RECFM=V). When the record format is not variable length, the SLI flag is also set to 1 if:

- The record format is not fixed-length records written as standard blocks (RECFM=FS or RECFM=FBS), and
- The data set is not a track-overflow data set (RECFM=T).

4 The read sector CCW is present if the device includes the rotational position sensing feature. Otherwise, the NOP CCW is present.

5 Each M/T read count CCW reads the count field of the next sequential record. (The next record might be on the same track, or it might be record one on the next track, next cylinder, or next extent.) When the user's program requests the next input record, the prolog's search CCW can locate it—minimizing the number of times search-previous logic is required.

When the BFTEK=R option is specified to read a BDAM data set and search-previous logic is required, an input channel program segment is built to read only the record's count field (needed for offset READ processing). This channel program segment is not chained to another input channel program segment.

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	M/T READ COUNT	SAMCNTS	C	8
CCW2	READ SECTOR[1]	SAMSECT	S	1
	NOP[1]	—	S	1

FLAGS

C = Chain command

S = SLI

Note:

1 The read sector CCW is present if the device includes the rotational position sensing feature. Otherwise, the NOP CCW is present.

Channel Program for Erase CCWs for BSAM Load Mode, Track Overflow (IGG0191M)

WITHOUT ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1[1]	SCH ID EQ	CCW10	C	5
CCW2	TIC	*-8		
CCW3	WRT DATA	CCW10	CS	7
CCW4	SCH ID EQ	CCW10	C	5
CCW5	TIC	*-8		
CCW6	RD DATA		CSK	8
CCW7	ERASE	CCW7	CS	8
CCW8	M/T RD HA		C	5
CCW9	TIC	CCW1		
CCW10	R0 ADDR = CCHH0000			

WITH ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1[1]	SET SECTOR	SECTOR=0	C	1
CCW2	SRCH ID EQ	CCW13	C	5
CCW3	TIC	*-8		
CCW4	WRT DATA	CCW13	CS	7
CCW5	SET SECTOR	SECTOR=0	C	1
CCW6	SCH ID EQ	CCW13	C	5
CCW7	TIC	*-8		
CCW8	RD DATA		CSK	8
CCW9	ERASE	CCW9	CS	8
CCW10	SET SECTOR	SECTOR=0	C	1
CCW11	M/T RD HA		C	5
CCW12	TIC	CCW1		
CCW13	R0 ADDR CCHH0000			

FLAGS

C = Command Chain

S = SLI

K = Skip

Note:

[1] Address of CCW1 is stored in DCBEOBW.

Channel Program for BSAM Load Mode, Track Overflow (IGG0191M)

WITHOUT ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	SCH ID EQ	IOBSEEK.+3.	C	5
CCW2	TIC	*-8		
CCW3	WRT CKD	IOBDNRCF(1)	D	8
CCW4				
CCW5[1]	TIC/NOP[2]	CCW20	CS	CCW20
CCW6	SCH ID EQ	IOBR0CNT(1)	C	5
CCW7	TIC	*-8		
CCW8	WRT DATA	IOBR0DAT(1)	C	8
CCW9[1]	READ R0		SK	16
CCW10[3]	M/T SCH ID EQ	IOBR0CNT(2)	C	5
CCW11	TIC	*-8		
CCW12	WRT CKD	IOBDNRCF(2)	D	8
CCW13				
CCW14[1]	TIC/NOP[2]	CCW19	CS	CCW19
CCW15	SCH ID EQ	IOBR0CNT(2)	C	5
CCW16	TIC	*-8		
CCW17	WRT DATA	IOBR0DAT(2)	C	8
CCW18[1]	READ R0 (WRITE CHECK)		SK	16
CCW19	SEEK CYL	IOBSEEK.+1.	C	6
CCW20	SCH ID EQ	IOBDNRCF(1)	C	5
CCW21	TIC	*-8		
CCW22	RD KD		SK	KL+DL

FLAGS

D = Data Chain

C = Command Chain

S = SLI

K = Skip

Notes:

[1] CCWs 5, 9, 14, and 18 are omitted if verify is not specified.

[2] The TIC/NOP at CCW5 and CCW14 is set to NOP if Record 0 is to be written on this track.

[3] CCWs 10 through 18 are repeated as many times as are needed to write all segments.

Channel Program for Create BDAM (IGG0199L)

WITHOUT ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	WRT CKD	IOBDNRCF	D	8
CCW4	WRT CKD	(WRITE CHECK)[1]	C	K+BLKSIZE.
CCW5	SCH ID EQ	IOBDNRCF	C	5
CCW6	TIC	*-8		
CCW7	RD KD		CSK	256
CCW8	SCH ID EQ	IOBROCNT	C	5
CCW9	TIC	*-8		
CCW10	WRT DATA	IOBRODAT	C	8
CCW11	SCH ID EQ[2]	IOBROCNT	C	5
CCW12	TIC	*-8		
CCW13	RD DATA		CSK	1
CCW14	ERASE[3]	*	S	8

FLAGS

D = Data Chain

C = Command Chain

S = SLI

K = Skip

Notes:

1 CCWs 5 through 7 are omitted if write check is not specified.

2 CCWs 11 through 13 are always generated for format-U and format-V records, or if write check is specified.

3 CCW14 is generated for format-U and format-V records only.

Channel Program for Create BDAM (IGG0199L)

WITH ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	WRT CKD	IOBDNRCF	D	8
CCW5	WRT CKD		C	KL+BLKSIZE
CCW6	RD SECTOR (WRITE CHECK)[1]	SECTOR2	C	1
CCW7	SET SECTOR	SECTOR2	C	1
CCW8	SCH ID EQ	IOBDNRCF	C	5
CCW9	TIC	*-8		
CCW10	RD KD		CSK	256
CCW11	SET SECTOR	SECTOR=0	C	1
CCW12	SCH ID EQ	IOBR0CNT	C	5
CCW13	TIC	*-8		
CCW14	WRT DATA	IOBR0DAT	C	8
CCW15	SET SECTOR[2]	SECTOR2	C	1
CCW16	SCH ID EQ	IOBR0CNT	C	5
CCW17	TIC	*-8		
CCW18	RD DATA		CSK	1
CCW19	ERASE[3]	*	S	8

FLAGS

D = Data Chain

C = Command Chain

S = SLI

K = Skip

Notes:

1 CCWs 7 through 10 are omitted if write check is not specified.

2 CCWs 15 through 18 are always generated for format-V and format-U records or if write check is specified.

3 CCW19 is generated for format-U and format-V records only.

Channel Program for BSAM Load Mode, Track Overflow (IGG0199M)

WITH ROTATIONAL POSITION SENSING

CCW#	COMMAND CODE	ADDRESS	FLAGS	COUNT
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	WRT CKD	IOBDNRCF(1)	D	8
CCW5	00			
CCW6[1]	TIC/NOP[2]	CCW24	CS	CCW24
CCW7	SET SECTOR	SECTOR=0	C	1
CCW8	SCH ID EQ	IOBR0CNT(1)	C	5
CCW9	TIC	*-8		
CCW10	WRT DATA	IOBR0DAT(1)	C	8
CCW11	SET SECTOR	SECTOR=0	C	1
CCW12	READ R0		SK	16
CCW13	M/T SCH ID EQ	IOBR0CNT(2)	C	5
CCW14	TIC	*-8		
CCW15	WRT CKD	IOBDNRCF(2)	D	8
CCW16				
CCW17[1]	TIC/NOP[2]	CCW24	CS	CCW24
CCW18	SET SECTOR	SECTOR=0	C	1
CCW19	SCH ID EQ	IOBR0CNT(2)	C	5
CCW20	TIC	*-8		
CCW21	WRT DATA	IOBR0DAT(2)	C	8
CCW22	SET SECTOR	SECTOR=0	C	1
CCW23	READ R0		SK	16
CCW24	RD SECTOR	SECTOR2	C	1
CCW25	SEEK CYL	IOBSEEK+1	C	6
CCW26	SET SECTOR	SECTOR1	C	1
CCW27	SCH ID EQ	IOBDNRCF(1)	C	5
CCW28	TIC	*-8		
CCW29	RD KD		SK	KL+DL

FLAGS

D = Data Chain

C = Command Chain

S = SLI

K = Skip

Notes:

1 CCWs 11, 12, 22, 23, and 25 through 29 are omitted if verify is not specified.

2 The TIC/NOP at CCW6 and CCW17 is set to NOP if Record 0 is to be written on this track.

INDEX

A

B

C

D

E

F

G

I

J

L

M

N

O

P

Q

R

S

T

U

W

Numerics

LY26-3967-0

Reader's Comment Form

MVS/XA SAM Logic

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Note: Staples can cause problems in automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

List TNLs here:

If you have applied any technical newsletters (TNLs) to this book, please list them here:

Last TNL _____

Previous TNL _____

Previous TNL _____

Fold on two lines, tape, and mail. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.) Thank you for your cooperation.

LY26-3967-0

Reader's Comment Form

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape | Please do not staple | Fold and tape

IBM

MVS/XA SAM Logic (File No. S370-30) Printed in U.S.A. LY26-3967-0

MVS/Extended Architecture
SAM Logic

Order No. LY26-3967-0
File No. S370-30

Printed in U.S.A.