SY26-3832-1
File No. S370-30

Systems

OS/VS2 SAM Logic

Release 3.8

IBM

Second Edition (February 1975)

This edition, as amended by technical newsletters SN26-0917, SN26-0931, SN26-0934, and SN26-0956, applies to Release 3.8 of IBM OS/VS2, and to any subsequent releases until otherwise indicated in new editions or technical newsletters.

The changes for this edition are summarized under "Summary of Amendments" following the list of diagrams. 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 periodically made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370 Bibliography,* GC20-0001, for the editions that are applicable and current.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming or services which are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming or services in your country.

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 reader's 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 any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

PREFACE

The information in 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.

A general understanding of data management is prerequisite knowledge for understanding the information in this book. See *OS/VS2 Data Management Services Guide,* GC26-3875, for background information on data management.

The manual is organized into the following sections:

"Introduction." This section contains a brief description of the sequential access method (SAM) routines and 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.

"Method of Operation." The SAM routines are described in the following categories:

1. Queued sequential access method (QSAM) routines that cause storage and retrieval of data records arranged in sequential order.
2. Basic sequential access method (BSAM) routines that cause storage and retrieval of data blocks arranged in sequential order.
3. 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.
4. Executors that operate with input/output support routines.
5. Buffer-pool management routines that furnish buffer space in virtual storage. Figure numbers appear as a running head in this section. The running heads identify figures that list functionally related groups of modules, appendages, or executors.
6. Problem Determination that assists the user in determining the causes of ABENDs by providing more information on the reason for the termination.
7. SVC routines that provide Supervisor state operation for functions that cannot be done in the problem state or in the user's key.
8. 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." This section contains diagrams that describe the organization and flow of control of the SAM routines.

"Directory." The 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 contain information about the module.

"Data Areas." This section shows how various control blocks are used in QSAM and BSAM. The access method save area for user totaling and the JES compatibility interface control block are also described. This section does not

describe in detail all fields of the system control blocks referred to in this manual. For information about system control blocks not included, see *OS/VS2 Data Areas.*

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

"Appendixes." These sections describe code conversion routines, BSAM/QSAM channel programs, update channel programs, chained scheduling channel programs, and BSAM channel programs.

Prerequisite Reading

For information about processing sequential and partitioned data sets:

- *OS/VS2 Data Management Services Guide,* GC26-3875

Related Reading

For specific information about the macro instructions required to process sequential and partitioned data sets:

- *OS/VS2 Data Management Macro Instructions,* GC26-3873

For specific information about Open, Close, and End-of-Volume routines:

- *OS/VS2 Open/Close/EOV Logic,* SY26-3827

For information about system control blocks:

- *OS/VS2 Data Areas,* SYB8-0606
- *OS/VS2 System Programming Library: Debugging Handbook,* GC28-0632

For information about the Job Entry Subsystem:

- *OS/VS2 JES2 Logic,* SY28-0622

For information about EXCP, EXCP appendages, DEVTYPE, DEBCHK, UCS images, FCB images, and IMGLIB:

- *OS/VS2 System Programming Library: Data Management,* GC26-3830

For information on the sequential access method routines used by the IBM 1287 and 1288 Optical Character Readers:

- *OS Data Management Macro Logic for IBM 1285/1287/1288,* GY21-0013

For information on the sequential access method routines used by the IBM 1419 Magnetic Character Reader:

- *OS BSAM Logic for IBM 1419/1275,* GY21-0012

For information about the IBM 3800 Printing Subsystem, character arrangement tables, writeable character generation modules, character sets, translate tables, graphic modification modules, copy modification modules, FCB images, and forms overlay negatives:

- *IBM 3800 Printing Subsystem Programmer's Guide,* GC26-3846

For information about the sequential access method routines used by the IBM 3886 Optical Character Reader:

- *OS/VS IBM 3886 Optical Character Reader Model 1 Logic,* SY24-5162

For information on system codes and messages:

- *OS/VS Message Library: VS2 System Messages,* GC38-1002
- *OS/VS Message Library: VS2 System Codes,* GC38-1008

For information on I/O Supervisor routines:

- *OS/VS2 I/O Supervisor Logic,* SY26-3823

For information on SAM and BPAM records written to SYS1.LOGREC:

- *OS/VS2 SYS1.LOGREC Error Recording Logic,* SY28-0678

For information on utilities used for problem determination and diagnosis:

- *OS/VS2 Service Aids Logic,* SY28-0643

For information on the sequential access method routines to support the IBM 3890 Document Processor:

- *OS/VS Logic for IBM 3890 Document Processor,* SY24-5163

CONTENTS

Preface ... 3
Prerequisite Reading ... 4
Related Reading ... 4

Illustrations ... 9
Figures ... 9
Diagrams ... 10

Summary of Amendments ... 11
Release 3 ... 11
Release 2 ... 12

Introduction ... 15

Method of Operation ... 17
Queued Sequential Access Method Routines ... 17
Get Routines ... 17
Simple-Buffering Get Routines ... 17
Parallel Input Processing Routine ... 32
Update Mode Get Routine ... 52
Put Routines ... 40
Simple Buffering Put Routines ... 40
Update Mode PUTX Routine ... 52
End-of-Block Routines ... 53
Ordinary End-of-Block Routines ... 54
Chained Channel-Program Scheduling End-of-Block Routines ... 64
Track-Overflow and User-Totaling Save Routines ... 74
Synchronizing-and-Error-Processing Routines ... 77
Appendages ... 87
End-of-Extent Appendages ... 88
Start I/O (SIO) Appendages ... 93
Channel-End Appendages ... 94
Program Controlled Interruption (PCI) Appendage (Execution of Channel Programs Scheduled by Chaining) ... 106
Abnormal-End Appendages ... 108
QSAM Control Routines ... 111
Basic Sequential Access Method Routines ... 114
Read and Write Routines ... 114
Check Routines ... 123
BSAM Control Routines ... 128
Basic Partitioned Access Method Routines ... 134
BPAM Routines ... 134
Dummy Data Set ... 135
Sequential Access Method Executors ... 135
DCB Relocation to Protected Work Area ... 135
Open Executors ... 136
Stage 1 Open Executors ... 136
Stage 2 Open Executors ... 148
Stage 3 Open Executors ... 163
Close Executors ... 172
Force Close Executors ... 177
Buffer-Pool Management ... 178
Problem Determination ... 183
SVC Routines ... 184

DEVTYPE Routine ... 185
IMGLIB Routine ... 185
Track Balance, Track Overflow Erase Routines ... 186
BSP Routine ... 187
STOW Routines ... 188
BLDL or FIND Routines ... 190
SYNADAF and SYNADRLS Routines ... 192
SETPRT and SETDEV Routines ... 197
Task Recovery Routines (TRR) ... 200.6

Program Organization and Flow of Control ... 207
Diagram A: Sequential Access Methods—Overview ... 207
Diagram B: QSAM Get and Put Routines ... 209
Diagram C: BSAM/BPAM Read/Write and Check Routines ... 211
Diagram D: Sequential Access Method Open Executors ... 213
Diagram E: SAM Flow of Control for Open Executors ... 215
Diagram F: QSAM Flow of Control ... 217
Diagram G: BSAM/BPAM Flow of Control ... 219
Diagram H: QSAM Flow of Control with EOV Routines ... 221
Diagram I: BSAM Flow of Control with EOV Routines ... 223
Diagram J: QSAM Operation with FEOV Routine ... 225
Diagram K: Open Processing for SAM Subsystem Interface Executors ... 227
Diagram L: Close Processing for SAM Subsystem Interface Executors ... 229
Diagram M: SAM Subsystem Interface Flow of Control for SYSIN/SYSOUT Data Sets ... 231
Diagram N: Force Close Processing ... 233
Diagram O: SYNADAF Flow of Processing ... 235

Directory ... 237

Data Areas ... 243
Message CSECT—IGGMSG ... 243
SETPRT Work Area (SPW)—IGGSPW ... 244
BLDL Work Area—SPW5 ... 244.2
Buffer Pool Control Block—IGGBCB ... 245
Parameter List—IGGPARML ... 246
SAM OPEN/CLOSE Work Area—IGGSCW ... 247
SAM/PAM/DAM GTRACE Buffer—IGGSPD ... 248
STOW Work Area—IGGSTW ... 248
SYNADAF General Registers Save Area and Message Buffer Area—IGGSYN ... 252
SETPRT Parameter List—IHASPP ... 254
Access Method Save Area for User Totaling ... 254.2

Diagnostic Aids ... 255
QSAM Control Blocks ... 255
BSAM Control Blocks ... 256
JES Compatibility Interface Control Block (CICB) ... 258
ABEND Codes and Cross-Reference Table ... 259
SETPRT Executor Return Codes and Messages (For 3800 Only) ... 262
SAM Register Saving Convention ... 262.4

Appendix A: Paper Tape Code Conversion Routines ... 263

Appendix B: BSAM/QSAM Channel Programs ... 265

Appendix C: Update Channel Programs ... 275

Appendix D: Chained Scheduling Channel Programs ... 281

Appendix E: BSAM (BDAM Create) Channel Programs ... 289

Index ... 295

ILLUSTRATIONS

Figures

Figure 1. Module Selector—Simple Buffering Get Modules 19
Figure 2. Order of Records Using Get Routines for Data Sets Opened for RDBACK (IGG019AM, IGG019AN) 25
Figure 3. The Two Parts of an Update Channel Program (Empty, Refill) 34
Figure 4. Relation of Seek Addresses in Three Successive QSAM Update Channel Programs 35
Figure 5. Module Selector—Update-Mode Get Modules 36
Figure 6. Module Selector—Simple Buffering Put Modules 42
Figure 7. Module Selector—Ordinary End-of-Block Modules 55
Figure 8. IOB SAM Prefixes for Normal and for Chained Scheduling 65
Figure 9. Module Selector—Chained Channel-Program Scheduling, End-of-Block Modules 66
Figure 10. Comparison of the IOB SAM Prefixes for Normal and for Chained Scheduling 66
Figure 11. Track-Overflow Records 74
Figure 12. Module Selector—Track-Overflow,75 End-of-Block Modules. 75
Figure 13. Module Selector—QSAM Synchronizing-and-Error-Processing Modules 79
Figure 14. Module Selector—Error-Processing Modules 85
Figure 15. Module Selector—Appendages 89
Figure 16. Module Selector—Control Modules 112
Figure 17. Control Routines that Are Expansions of Macro Instructions 112
Figure 18. Module Selector—Read and Write Modules 115
Figure 19. Module Selector—Check Modules 124
Figure 20. Module Selector—Control Modules Selected and Loaded by the Open Executor 129
Figure 21. Control Routines that are Expansions of Macro Instructions . 129
Figure 22. BPAM Routines Residence 134
Figure 23. Sequential Access Method Executors—Control Sequence 135
Figure 24. Open Executor Selector—Stage 1 138
Figure 25. Open Executor Selector—Stage 2 149
Figure 26. Open Executor Selector—Stage 3 165
Figure 27. Close Executor Selector 172
Figure 28. Buffer-Pool Management Routines 178
Figure 29. Buffer-Pool Control Block 179
Figure 30. GETPOOL Buffer-Pool Structures 179
Figure 31. Build Buffer-Structuring Table 180
Figure 32. Build Buffer-Pool Structure 180
Figure 33. Logical Record Buffer-Pool Control Block 181
Figure 34. Record Area 182
Figure 34.1 SETPRT Executor Selector 198
Figure 35. Access Method Save Area for User Totaling 254.2
Figure 36. QSAM Control Blocks 255
Figure 37. BSAM Control Blocks 257
Figure 38. Control Block Structure for SYSIN/SYSOUT Data Sets 258

Diagrams

Diagram A. Sequential Access Methods—Overview 207
Diagram B. QSAM Get and Put Routines 209
Diagram C. BSAM/BPAM Read/Write and Check Routines 211
Diagram D. Sequential Access Method Open Executors 213
Diagram E. SAM Flow of Control for Open Executors 215
Diagram F. QSAM Flow of Control 217
Diagram G. BSAM/BPAM Flow of Control 219
Diagram H. QSAM Flow of Control with EOV Routines 221
Diagram I. BSAM Flow of Control with EOV Routines 223
Diagram J. QSAM Operation with FEOV Routine 225
Diagram K. Open Processing for SAM Subsystem Interface Executors 227
Diagram L. Close Processing for SAM Subsystem Interface Executors 229
Diagram M. SAM Subsystem Interface Flow of Control for SYSIN/SYSOUT Data Sets 231
Diagram N. Force Close Processing 233
Diagram O. SYNADAF Flow of Processing 235

SUMMARY OF AMENDMENTS

November 1979

OS/VS2 MVS 3800 Printing Subsystem Enhancements

This programming enhancement, applicable to OS/VS2 MVS Release 3.8, supports new 3800 Printing Subsystem installed changes and improved software support:

- Improved paper repositioning operator control
- Load command changes
- Enhanced SETPRT messages and return codes
- Support for paper jam and cancel key during SETPRT
- Common interface to user for schedule for print and setup of SYSOUT data sets
- User control of WCGM management
- Multiple extents for SYS1.IMAGELIB

July 1979

Device Support Enhancements

Provides a routine, invoked by a new system macro, TRKCALC, to perform track capacity calculations. Using this macro will allow track capacity algorithm independence.

June 1979

New Devices

The IBM 3203 Printer, Model 5, is supported as a Model 4.

OS/VS2 Data Management Support for Mass Storage System (MSS) Extensions Program Product

New Programming Support

MSS Extensions program product is supported with this programming support.

August 1978

This technical newsletter incorporates and replaces all previous SU information in this publication.

New Devices

IBM 3203 Printer, Model 4

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" section 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 both 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" section lists the macro statements that are used with sequential access method programming techniques. The chart 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 track-overflow and 3211 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 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 (providing 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 *OS/VS Data Management Services Guide.*

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. 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. 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 desired, the processing program issues a RELSE macro instruction. Control passes to a RELSE routine which sets an EOB condition. When records are spanned, one or more blocks can be skipped to find the first record in a new block.

The Open executor primes (that is, schedules for filling) the buffers if QSAM is used with a DCB opened for input, update, or readback. For the locate mode, all buffers except one are primed; for the move mode, all buffers are primed. The Open executor also sets an end-of-block condition; the first time that a Get routine gains control, it processes this condition in the usual way.

Upon return from the synchronizing-and-error-processing routine, the Get routines, which may be loaded for tape data sets, tests 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 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.

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

Access Method Options	**Selections**																	
INPUT, Get	X	X	X	X	X	X	X	X					X	X	X	X	X	X
RDBACK, Get									X	X	X	X						
Locate operating mode	X	X	X						X	X				X	X			
Move operating mode				X	X	X	X	X			X	X	X			X		
Data operating 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 specified on DD statement																		X
Card reader, only a single, buffer CNTRL							X	X										
Character conversion for paper tape													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						
IGG019AT[1]													AT					
IGG019BO														BO				
IGG019DJ																		DJ
IGG019FB															FB			
IGG019FD																FD		
IGG019FF																	FF	

1. This module also includes the character-conversion and synchronizing-and-error-processing routine for paper-tape devices.

Figure 1. Module Selector—Simple Buffering Get Modules

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.
- 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.

- 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.

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

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

RDBACK

and the DCB specifies:

Get

Simple buffering

Move operating mode

Fixed-length (unblocked, blocked, or blocked standard) or undefined-length record format

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

The Get routine operates as follows:

- It receives control when a GET macro instruction is encountered in a processing program.
- It tests for an EOB condition.
- If no EOB condition exists, it moves the next record to the work area, and updates the DCBRECAD field by reducing it by the value of the DCBLRECL field.

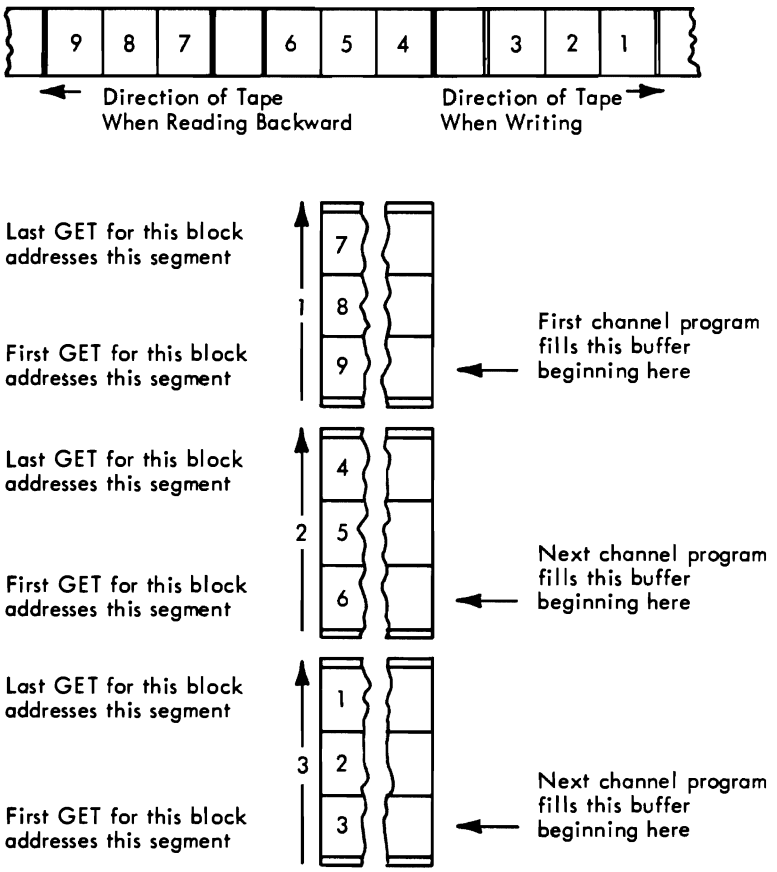


Figure 2. Order of Records Using Get Routines for Data Sets Opened for RDBACK (IGG019AM, IGG019AN)

- If an EOB condition exists, it issues a BALR instruction to obtain a new buffer through the input-synchronizing-and-error-processing routine, module IGG019AQ. The Get routine then moves the last record of the new buffer to the work area.
- It tests for a new EOB condition.
- If no new EOB condition exists, it returns control to the processing program.
- If a new EOB condition exists, it issues a BALR instruction to pass the present buffer to the end-of-block routine and then returns control to the processing program.

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

Figure 2, described for Get module IGG019AM, also illustrates the ordering of records using this module.

Get Module IGG019AT (Paper Tape): Module IGG019AT converts paper tape characters into EBCDIC characters and moves them to the work area. The Open executor selects and loads this module and one of the code conversion modules (see "Appendix A: Paper Tape Code Conversion Routines") if the Open parameter list specifies:

Input

and the DCB specifies:

Get

Simple buffering

Move operating mode

Paper-tape-character-conversion

The module consists of a Get routine and a character-conversion and synchronizing-and-error-processing routine.

The Get routine operates as follows:

- It receives control when a GET macro instruction is encountered in a processing program.
- It converts the next character and moves it to the work area.
- It continues converting and moving until one of the following conditions is met, resulting in stated effect:

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

  An EOB condition is encountered: The routine passes control to the end-of-block routine to refill the buffer, and then enters the character-conversion and synchronizing-and-error-processing routine to obtain a new buffer.

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

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

  A paper tape reader-detected error character is encountered: The routine moves the character to the work area without conversion and enters the character-conversion and synchronizing-and-error-processing routine.
- If one of the characters in the buffer is an undefined character, the module converts it to the hexadecimal character FF, moves it to the work area, and continues conversion. When one of the previous conditions is met, control passes to the character-conversion and synchronizing-and-error-processing routine.

The character-conversion and synchronizing-and-error-processing routine operates as follows:

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

- For the Accept-error option, the routine returns control to the processing program.
- For the Skip-error option, the routine fills the work area again.
- For the ABE-error option, or if no error option is specified, the routine issues the ABEND macro instruction.

The modules containing the tables used for code conversion are listed in "Appendix A: Paper Tape Code Conversion Routines."

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 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 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 IGG019AO). 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 that is described in the section, "Simple Buffering Put Routines" (see Figure 6). 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 RPL (request parameter list). 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 RDW (record descriptor word) in the first four bytes of the work area.
- It passes control to the JES (job entry subsystem) 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 the 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 13) for EODAD and concatenation processing
- If an error condition is detected, control is passed to the error-processing module, IGG019AH (see Figure 14). If control is returned to this routine, the GET request is reissued if DCB EROPT is SKIP. 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 not do any 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 1s 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.

- 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 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 way. The procedure on 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.
- 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 PDAB (parallel data address block) 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 four 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.

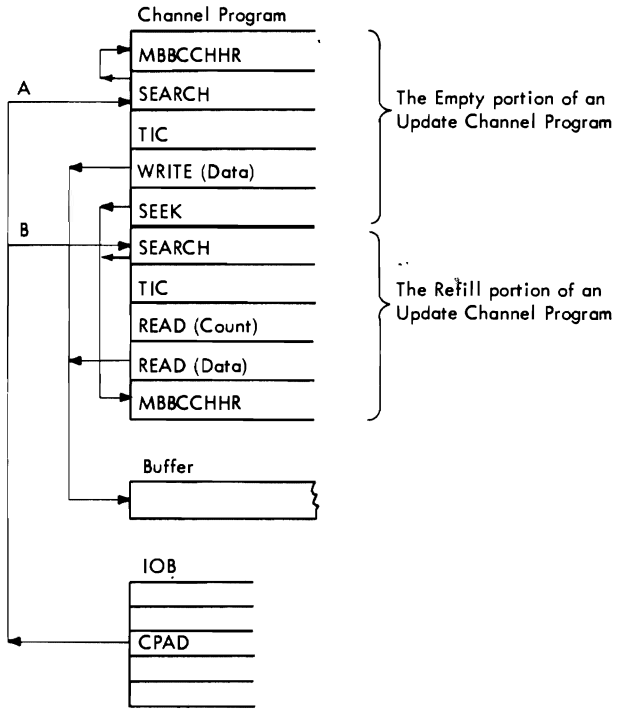
The first three of these determinations are made at every pass through the routine. The last determination is made after the routine establishes that the buffer can be scheduled.

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 is determined by whether an end-of-block condition exists. In the update mode, one determination that an end-of-block condition exists causes both the last buffer to be scheduled and a new buffer to be sought. An end-of-block condition exists for unblocked records at every pass through the routine; for blocked records it exists if the values in the DCBRECAD (the address of the current record) 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 IOB to point to the beginning of either one of two parts (empty and refill) of the QSAM update channel program. Empty



Legend:

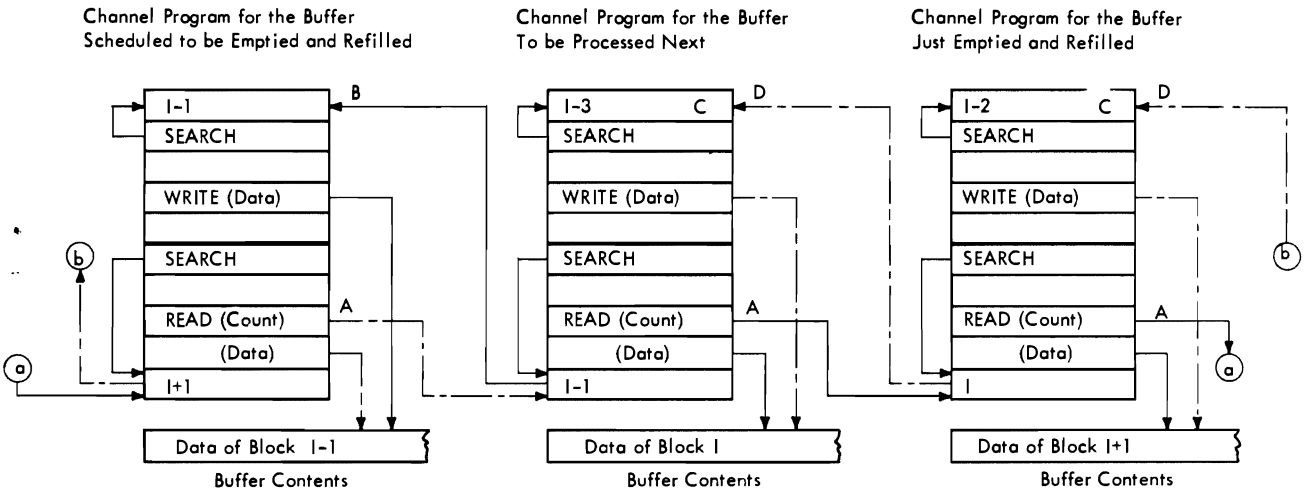
A - Address of channel program (CPAD) used to empty and refill the buffer. (A PUTX macro-instruction was addressed to a record in this buffer.)

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

Figure 3. The Two Parts of an Update Channel Program (Empty, Refill)

writes out of the buffer and reads into that same buffer (see Figure 3). If execution of a QSAM update channel program begins with the empty part, execution of the refill part always follows. Each part of the QSAM update channel program addresses a different location in auxiliary storage: the empty part addresses the location from which the block to be updated was read; the refill part addresses the location from which the last block was read. Addressing the last known block and skipping over its data field leads to the beginning of the next block, regardless of its address. This method of addressing a Search command to the block read previously to address a Read (count, key, and data) command to the next block is known as the search-previous technique. It makes the count field of the present block being read the Seek address of the refill portion of the next channel program. When a buffer is to be emptied (back to the original location of the block in auxiliary storage), the update mode Get routine obtains the block address from the Seek address of the refill part of the next channel program. It copies the address so that it becomes the Seek address for the empty part of the present channel program (see Figure 4). For a description of the processing for a refill-only QSAM update channel program, refer to the description of the update SIO appendage.

Whether to schedule the buffer for empty-and-refill or for refill-only depends on whether the block is to be updated. If the block is to be updated, the PUTX routine will have set the update flag on in the IOB; otherwise, the flag is off. To schedule the buffer for empty-and-refill, the Get routine sets the IOB to point to the empty portion of the channel program and obtains the



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

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

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

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

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

Figure 4. Relation of Seek Addresses in Three Successive QSAM Update Channel Programs

Seek address of the block to be updated from the refill portion of the next channel program. To schedule the buffer for refill-only, the Get routine sets the IOB to point to the refill portion of the channel program. The end-of-block condition which triggers this processing also causes control to pass to the end-of-block routine, module IGG019CC, for issuing the EXCP macro instruction and to the update-synchronizing-and-error-processing routine, module IGG019AF, for obtaining the next buffer.

The PUTX routine sets the update flag in the IOB and returns control to the processing program. The RELSE routine sets an end-of-block condition and returns control to the processing program.

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

Figure 5 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

1. This module also carries the Update-Mode PUTX routine.

Figure 5. Module Selector—Update-Mode Get Modules

Get Module IGG019AE: Module IGG019AE presents the processing program with the next input record, flags the IOB if the block is to be updated (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

With the rotational position sensing (RPS) feature, the new CCWs are bypassed when necessary.

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, it tests whether the buffer is to be emptied and refilled or is to be refilled only.
- If it is to be refilled only, it sets the IOB to point to the start of the Read portion of the update channel program and passes control to the end-of-block routine to cause scheduling of the buffer.

- If it is to be emptied and refilled, it sets the IOB to point to the start of the update channel program. The routine obtains the auxiliary storage address to be used by the Write portion of the channel program by copying the address used by the Read portion of the channel program associated with the next IOB. The routine then passes control to the end-of-block routine to cause scheduling of the buffer.
- On return of control from the end-of-block routine, the Get routine passes control to the update-synchronizing-and-error-processing routine, module IGG019AF, to obtain a new full buffer.
- On return of control from the synchronizing routine, the Get routine updates the DCBLRECL field, 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

Logical record interface

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 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 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 address used by the Read portion of the channel program 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, it tests whether the buffer is to be emptied and refilled or is to be refilled only.
- If it is to be refilled only, it sets the IOB to point to the start of the read portion of the update channel program and passes control to the EOB routine to cause scheduling of the buffer.

- If it is to be emptied and refilled, it sets the IOB to point to the start of the update channel program. The routine obtains the auxiliary storage address to be used by the Write portion of the channel program by copying the address used by the Read portion of the channel program associated with the next IOB. The routine then passes control to the EOB routine to cause scheduling of the 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.
- It sets the IOB to point to the start of the read portion of the update channel program and passes control to the EOB routine to cause scheduling of the 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 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 spanned-record 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.

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 6. For update mode, the PUTX routine resides in the Get module for update mode. See Figure 5 (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.

Simple Buffering Put Routines

Simple buffering Put routines use buffers whose ending address and the address of the next or current record is 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 five or more bytes remain in the buffer. When fewer than five bytes remain in the buffer, an EOB condition occurs.

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 6 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.

Access Method Options	**Selections**										
Output, Put/PUTX	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	
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 6. Module Selector—Simple Buffering Put Modules

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

Output

and the DCB specifies:

Put

Simple buffering

Locate operating mode

Fixed-length (unblocked, blocked or blocked standard) or undefined-length record format

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

The Put routine operates as follows:

- It receives control when a PUT macro instruction is encountered in a processing program.
- It determines the address of the next buffer segment using the value in the DCBLRECL field.
- It tests for an EOB condition to determine whether a buffer is full and ready for emptying and 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 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.
- 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 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.

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 routine (see Figure 1).

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 decrementing the record length by four.

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)	DCBLRECL

- 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.
- It passes control to the job entry subsystem 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 14).
- 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 not do any 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 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 five 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 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 five 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 Routine

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

There is one update mode Put routine; it is part of module IGG019AE, which is described under "Update Mode Get Routine" (see Figure 5).

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 are 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↑	Unchanged
3	IOB-8 (or ICB)	Unchanged
4-6	N/A	Not restored
7	Read or Write CCW offset	Unchanged
8	Caller's base address	Unchanged
9-10	User's registers	Restored*
11-12	User's registers	Unchanged
13	Save area	Unchanged
14	Caller's return address	Unchanged
15	Entry point address	Not restored

*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 to the I/O supervisor, except when a channel program is chained to another one not yet executed. End-of-block routines provide device-oriented entries for the channel program, such as control characters and auxiliary storage addresses.

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, except when it is used with an output data set with track overflow.

- Chained channel-program scheduling end-of-block routines. These routines perform device-oriented processing and attempt to chain channel programs when chained channel-program scheduling is used.
- Track-overflow, end-of-block routine. This routine performs device-oriented processing. It computes segment lengths and constructs count fields when track overflow, which uses normal channel-program scheduling, is used with an output data set.
- User-totaling routine. This routine moves the contents of the user's totaling area to the user-totaling save area pointed to by the DEB.

Ordinary End-of-Block Routines

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

Figure 7 lists the routines available and the For 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, for example, shows that when normal channel-program scheduling is used and the device type is magnetic tape, routine IGG019CC is selected and loaded for use as the end-of-block routine for that DCB.

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.

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

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

The data set is opened for Input, and the DCB specifies normal channel-program scheduling, direct-access storage device, and a record format other than fixed-length standard.

The data set is opened for INOUT or OUTIN, and the DCB specifies normal channel-program scheduling, direct-access storage device and a record format other than fixed-length standard. The address of this module is placed in the DCBEOBR field.

The data set is opened for Update.

Access Method Options	Selections																							
Normal channel program scheduling	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Input, or					X	X				X	X													
Update				X				X																
Output, or									X		X	X	X											
INOUT, OUTIN					X				X	X	X	X								X				
Card reader or paper tape reader	X	X																						
Printer or Card Punch													X	X	X	X	X	X	X		X			
Print (3525)																						X	X	
Interpret Punch (3525)																								X
Data Protection Image (3525)																					X			
Magnetic tape			X				X																	
Direct-access storage					X				X	X	X	X												
Track Overflow						X																		
Record format not fixed-length standard					X				X															
Record format is fixed-length standard										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 card[1]																					X			
User totaling facility							X	X	X			X	X											
Associated Data Set (3525)		X													X	X		X		X		X		
IGG019AX[2]							AX	AX	AX			AX	AX											
IGG019CC	CC	CC	CC	CC	CC	CC																		
IGG019CD										CD	CD													
IGG019CE													CE	CE	CE	CE								
IGG019CF																	CF	CF						
IGG019CT[1]																			CT					
IGG019FK																				FK				
IGG019FQ																						FQ	FQ	
IGG019FU																								FU
IGG019TC							TC	TC	TC															
IGG019TD												TD	TD											

1. When either of these LABEL subparameters 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.

2. This module is described later in this section under "Track-Overflow and User-Totaling Save Save Routines."

Figure 7. Module Selector—Ordinary End-of-Block Modules

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 blocksize, whichever is less; however, with the ASCII feature, format-D records are padded with the ASCII padding character, X'5F', instead of the 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 the device type is direct access and more than one IOB is associated with the DCB, the module does the following:
  a. It checks for a cylinder change in the IOBSEEK field in the next IOB by comparing it to the cylinder value in the DCBFDAD field in the DCB.
  b. It copies the IOBSEEK field in the next IOB into the DCBFDAD field in the DCB.
  c. If a change in cylinder value was found and the new 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.
  d. It issues an EXCP macro instruction and returns control to the GET or READ routine.
- If a 3525 associated data set is being used, a test is made to determine the status of the Read-sequence flag.
  a. If the Read-sequence flag (DCBQSWS field) is on and the associated data set is not Read and Print, a WTP message is issued, which indicates that either the Get or Read sequence is invalid. 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.
  b. Tests are made to determine if the associated data set is either Read, Punch, and Print, or Read and Punch.
  c. 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.)
  d. 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.

e. A test is made to determine the status of the Print-sequence flag (DCBQSWS).

f. 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.

- If the device type is direct access and only one IOB is associated with the DCB, the module does the following:

  a. It 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.

  b. It copies the CCHHR portion of the DCBFDAD field in the DCB into the CCHHR portion of the IOBSEEK field in the IOB.

  c. If a change in cylinder value was found and the new cylinder value is on a page boundary (evenly divisible by 8), the DEBFLGS2 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.

  d. It issues an EXCP macro instruction and returns control to the GET or READ routine.

End-of-Block Module IGG019CD: Module IGG019CD schedules a channel program after determining that the next block fits on a track within the allocated extents.

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

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

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

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

If the record format is fixed-length standard, the address of this module is placed in both the DCBEOBR and DCBEOBW fields.

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.
- It calculates the block length using the overhead value for the last record. (This value is found in the resident I/O device table. The address of the table is in the DCBDVTBL 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 finds the next track as follows:

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

  It adds 1 to the value of TT.

  It passes control to the IECPCNVT routine, which converts the relative address of the next track into the full device address.
- If there is another track in the allocated extents, its full address has been entered in the DCBFDAD field and the block fits on the track.
- If there is no other track in the allocated extents (as shown by the error return code from IECPCNVT routine), an EOV condition exists. The module sets the DCBCIND1 field in the DCB and the CSW field in the IOB to show this, and returns control to the Get, Put, Read, or Write routine without issuing an EXCP macro instruction. The EOV condition is eventually recognized and processed in QSAM by the synchronizing routine and in BSAM by the Check routine.
- When the module determines that the block fits on the track, the module calculates the actual block length, using the overhead value for other than the last record. (This value is found in the resident I/O device table.) It adjusts the value in the DCBTRBAL field by this amount and updates the DCBFDAD field and the ID field of the count area of the block which is located immediately after the channel program. If the updated DCBFDAD value 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. The module then issues an EXCP macro instruction and returns control to the Get, Put, Read, or Write module.

End-of-Block Module IGG019CE: Module IGG019CE, if necessary, modifies channel programs for unit record output devices when ANS 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 a 3800 printer 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.

  a. 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.

  b. 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.

  c. 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.

  d. A test is made to determine the status of the Print-sequence flag.

e. If the Print-sequence flag is on, proper sequencing continues. If it is off, modules IGG019CE and IGG019CF continue with their normal functions.

f. 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 (ANS) 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 (CCW) which precedes the Write CCW (or the Select CCW, if the device is a 3800 printer with OPTCD=J specified).
- If the device is a 3800 printer 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.

  a. 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.

  b. 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.

c. 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.

d. A test is made to determine the status of the Print-sequence flag (DCBQSWS).

e. If the Print-sequence flag is on, proper sequencing continues. If it is off, modules IGG019CE and IGG019CF continue with their normal functions.

f. 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 card specifies LABEL=(,,,IN)

or

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

  The DD card 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 Printer. 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 ANS 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 which indicates 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 ANS and machine control characters. If ANS 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 ANS 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 his data set and if one of the following conditions exists:

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

The data set is opened for Output, and the DCB specifies normal channel-program scheduling, direct-access storage device, and a record format other than fixed-length standard.

The data set is opened for INOUT or OUTIN, and the DCB specifies normal channel-program scheduling, direct-access storage device and a record format other than fixed-length standard. The address of this module is placed in the DCBEOBR field.

The data set is opened for Update.

The module operates as follows:

- It receives control when a 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 device type is magnetic tape, 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.

- If the device type is direct access and more than one IOB is associated with the DCB, the module does the following:

  a. It checks for a cylinder change in the IOBSEEK field in the next IOB by comparing it to the cylinder value in the DCBFDAD field in the DCB.

  b. It copies the IOBSEEK field in the next IOB into the DCBFDAD field in the DCB.

  c. If a change in cylinder value was found and the new 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.

  d. It issues an EXCP macro instruction and returns control to the GET or READ routine.

- If the device type is direct access and only one IOB is associated with the DCB, the module does the following:

  a. It 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.

  b. It copies the CCHHR portion of the DCBFDAD field in the DCB into the CCHHR portion of the IOBSEEK field in the IOB.

  c. If a change in cylinder value was found and the new cylinder value is on a page boundary (evenly divisible by 8), the DEBFLGS2 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.

  d. It issues an EXCP macro instruction and returns control to the GET or READ routine.

End-of-Block Module IGG019TD: Module IGG019TD schedules a channel program after determining that the next block fits on a track within the allocated extents.

The Open executor selects and loads this module if the user specified the user totaling facility (that is, if bit 6 is 1 in DCBOPTCD) for his data set and if one of the following conditions exists:

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

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

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

The module operates as follows:

- It receives control when a Get or a Put routine finds that a buffer is ready to be scheduled, or at the conclusion of the processing performed by a Read or Write routine.
- It 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 calculates the block length using the overhead value for the last record. (This value is found in the resident I/O device table. The address of the table is in the DCBDVTBL 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 finds the next track as follows:

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

  It adds 1 to the value of TT.

  It passes control to the IECPCNVT routine, which converts the relative address of the next track into the full device address.
- If there is another track in the allocated extents, its full address has been entered in the field DCBFDAD and the block fits on the track.
- If there is no other track in the allocated extents (as shown by the error return code from routine IECPCNVT), an EOV condition exists. The module sets the DCBCIND1 field in the DCB and the CSW field in the IOB to show this, and returns control to the Get, Put, Read, or Write routine without issuing an EXCP macro instruction. The EOV condition is eventually recognized and processed—in QSAM by the synchronizing routine and in BSAM by the Check routine.
- When the module determines that the block fits on the track, the module calculates the actual block length, using the overhead value for other than the last record. (This value is found in the resident I/O device table.) It adjusts the value in the DCBTRBAL field by this amount, and updates the DCBFDAD field and the ID field of the count area of the block (located immediately after the channel program). If the updated DCBFDAD value 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. The module then issues

an EXCP macro instruction and returns control to the Get, Put, Read, or Write module.

Chained Channel-Program Scheduling End-of-Block Routines

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 and mainly uses the input/output block (IOB); disconnecting and posting is performed by appendages and uses the interruption control block (ICB). (For a description of the disconnecting process, refer to the program controlled interruption (PCI) appendages.) The IOB constructed by the Open executor when chained channel-program scheduling is used differs from the IOB used in normal channel-program scheduling. These differences are illustrated in Figure 8 and tabulated in Figure 10.

These routines join channel programs so that the channel executes successive channel programs without interruption as if they were one continuous channel program. To join the present channel program to one already scheduled, the end-of-block routine finds the last CCW of the preceding channel program by referring to the IOB and changes that CCW from a NOP command to a TIC command. If this joining is performed before the channel attempts to execute (more precisely, before it fetches) that CCW, the joining process is successful. If the execution of the preceding channel program is completed while the routine is operating, the joining is unsuccessful.

The routine tests the success or failure of the joining by testing whether the IOB has been posted as completed. If the IOB is not posted as completed, control is returned to the calling program. If the IOB is posted, the routine tests the ICB for the current channel program. If completed, control returns to the calling program; if not completed, the routine resets the IOB for the EXCP macro instruction and passes control to the I/O supervisor.

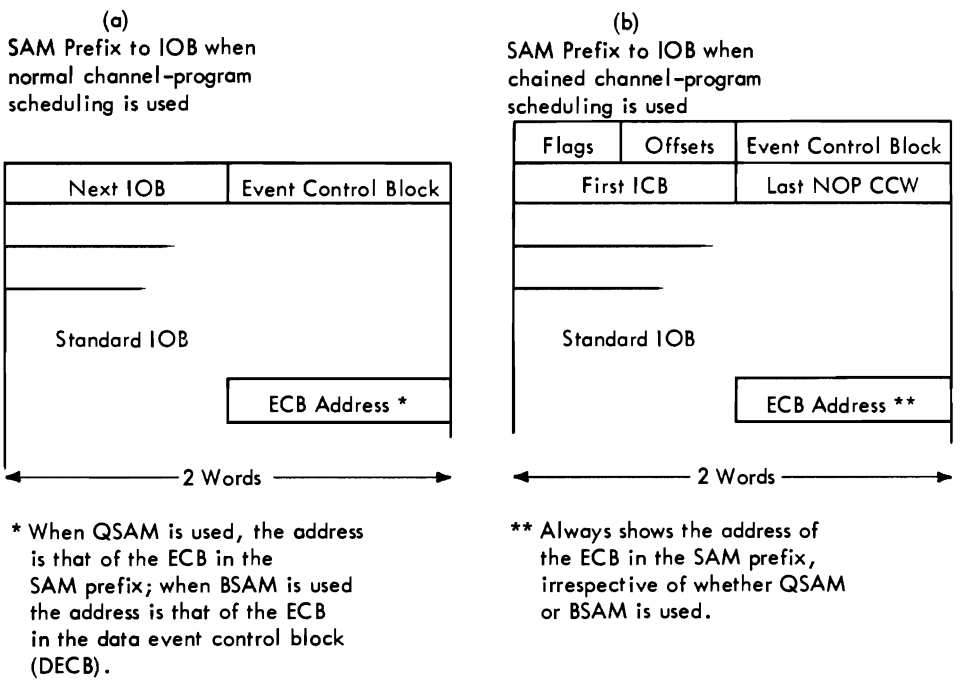


Figure 8. IOB SAM Prefixes for Normal and for Chained Scheduling

The chained scheduling end-of-block routines, like the ordinary end-of-block routines, provide device-oriented entries for channel programs. For direct-access devices they compute auxiliary storage addresses; for unit-record devices they process control characters. (No processing is performed for the PRTOV macro instruction since it and chained scheduling are mutually exclusive.) There are six 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 9 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 9 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	X	X	X
Input, or	X	X								
Output			X	X		X	X	X	X	X
Card reader	X									
Printer or card punch								X	X	X
Magnetic tape		X	X	X						
Direct-access storage					X	X	X			
No control character								X		
Machine control character									X	
ANS control character										X
User-totaling facility				X			X			
End-of-Block Modules										
IGG019AX[1]				AX			AX			
IGG019CV						CV				
IGG019CW	CW	CW	CW		CW					
IGG019CX								CX	CX	
IGG019CY										CY
IGG019TV							TV			
IGG019TW				TW						

1. This module is described later in this section under "Track-Overflow and User-Totaling Save Routines."

Figure 9. Module Selector—Chained Channel-Program Scheduling, End-of-Block Modules

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

Figure 10. Comparison of the IOB SAM Prefixes for Normal and for Chained Scheduling

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

Output

and the DCB specifies:

Chained channel-program scheduling

Direct-access storage

The module operates as follows:

- It receives control from a Put routine when that routine finds that a buffer is ready to be scheduled, or from a Write routine at the conclusion of its processing.
- It calculates the block length using the overhead value for a last block on a track. (This value is found in the resident I/O device table. The address of the table is in the DCBDVTBL field.) It compares the calculated block length with the value in the DCBTRBAL field of the DCB.
- If the block is equal to or less than the DCBTRBAL field value, the module determines that the block fits on the track.
- If the block length exceeds the DCBTRBAL field value, the module calculates the next sequential track address and compares it with the end address of the current extent shown in the data extent block (DEB).
- If no end-of-extent condition exists, it determines that the block fits on the track.
- If an end-of-extent condition exists, it seeks a new extent in the DEB.
- If a new extent exists, it updates the DCBFDAD and DCBTRBAL fields and determines that the block fits on the track.
- If there is no further extent, an EOV condition exists. The module sets the DCBCIND1 field in the DCB and the CSW field in the IOB to show end-of-volume, and returns control to the Get, Put, Read, or Write routine without issuing an EXCP macro instruction. The EOV condition is eventually recognized and processed—in QSAM by the synchronizing routine, 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 ID field of the count area of the block located immediately after the channel program.
- If the ICBSEEK value 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 the block fits on the track, the module next attempts to join the channel program for the current buffer to the preceding channel program (that is, chain schedule) by:

  Setting the ICB to not-complete.

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

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

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

- It determines whether the joining was successful by testing the ECB (pointed to by the IOB) to see if the I/O supervisor has posted the I/O event as completed.
- If the I/O supervisor did not post the event as completed, the joining was successful and the routine returns control to the calling routine.
- If the I/O supervisor did post the event as completed, the routine tests the ICB for the present channel program to find whether the joining was successful or not.
- If the present ICB remains unposted, the present channel program was not joined to the preceding one. The routine prepares to cause restart of the channel by copying the Seek address and the channel program start address from the current ICB into the IOB and uses the EXCP macro instruction to schedule the channel program. It then returns control to the calling routine.
- If the present ICB is posted as completed, the present channel program was joined successfully. (The routine was interrupted long enough, between the joining and the testing, for the channel program to be executed and for the channel-end appendage to post the ICB.) The routine returns control to the calling routine.

End-of-Block Module IGG019CW: Module IGG019CW attempts to join the present channel program to the last one in the chain of scheduled channel programs. If ASCII is used, the entire output buffer is translated from EBCDIC to ASCII. 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.

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 blocksize, whichever is less; however, with the ASCII feature, format-D records are padded with the ASCII padding character, X'5F', instead of zeros.
- If the device type is direct access, the module does the following:
  a. It checks for a cylinder change in the ICBSEEK field in the next ICB by comparing it to the cylinder value in the DCBFDAD field in the DCB.
  b. It copies the ICBSEEK field in the next ICB into the DCBFDAD field in the DCB.
  c. If a change in cylinder value was found and the new 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.
- The module attempts to join the channel program for the current buffer to the preceding channel program (that is, chain schedule) by:

  Setting the ICB to not-complete.

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

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

  Updating the SAM IOB prefix block to point to the end of the current channel program.
- It determines whether the joining was successful by testing the ECB (pointed to by the IOB) for a completion posting by the I/O supervisor.
- If the I/O supervisor did not post the event as completed, the joining was successful and the routine returns control to the calling routine.
- If the I/O supervisor did post the event as completed, the routine tests the ICB for the present channel program to find whether the joining was successful or not.
- If the present ICB remains unposted, the present channel program was not joined to the preceding one. The routine prepares to cause restart of the channel by copying the channel program Start address (and the Seek address, if direct-access storage) from the current ICB into the IOB, and uses the EXCP macro instruction to cause scheduling of the channel program. It then returns control to the calling routine.
- If the present ICB is posted as completed, the present channel program was joined successfully. (The routine was interrupted long enough, between the joining and the testing, for the channel program to be executed and for the channel-end appendage to post the ICB.) The routine returns control to the calling routine.

End-of-Block Module IGG019CX: Module IGG019CX, if necessary, modifies channel programs for unit-record output devices when 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, 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 printer 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.

- It determines whether the joining was successful by testing the ECB (pointed to by the IOB) to see if the I/O supervisor has posted the I/O event as completed.
- If the I/O supervisor did not post the event as completed, the joining was successful and the routine returns control to the calling routine.
- If the I/O supervisor did post the event as completed, the routine tests the ICB for the present channel program to find whether the joining was successful or not.
- If the present ICB remains unposted, the present channel program was not joined to the preceding one. The routine prepares to cause restart of the channel by copying the channel program Start address from the current ICB into the IOB, and uses the EXCP macro instruction to cause scheduling of the channel program. It then returns control to the calling routine.

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 (or the Select CCW, if the device is a 3800 printer with OPTCD=J specified).
- If the device is a 3800 printer 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.
- It determines whether the joining was successful by testing the ECB (pointed to by the IOB) to see if the I/O supervisor has posted the I/O event as completed.
- If the I/O supervisor did not post the event as completed, the joining was successful and the routine returns control to the calling routine.
- If the I/O supervisor did post the event as completed, the routine tests the ICB for the present channel program to find whether the joining was successful or not.
- If the present ICB remains unposted, the present channel program was not joined to the preceding one. The routine prepares to cause restart of the channel by copying the channel program Start address from the current ICB into the IOB, and uses the EXCP macro instruction to cause scheduling of the channel program. It then returns control to the calling routine.

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

End-of-Block Module IGG019TV: Module IGG019TV computes from the track balance (and from further allocated extents on this volume, if necessary) a valid storage address for a channel program for an output data set on a direct-access device and attempts to join the channel program to the preceding one. The Open executor selects and loads this module if the user specified the user-totaling option (that is, if bit 6 is 1 in DCBOPTCD) for his data set and if the Open parameter list specifies:

Output

and the DCB specifies:

Chained channel-program scheduling

Direct-access storage

The module operates as follows:

- It receives control from a Put routine that finds 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.
- 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 DCBDVTBL 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 ID field of the count area of the block located immediately after the channel program.

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

  Setting the ICB to not-complete.

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

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

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

- It determines whether the joining was successful by testing the ECB (pointed to by the IOB) to see if the I/O supervisor posted the I/O event as completed.

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

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

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

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

End-of-Block Module 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 his data set and if either of the following conditions exists:

The Open parameter list specifies Output and the DCB specifies chained channel-program scheduling and a direct-access device.

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

The module operates as follows:

- It receives control from a 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.
- If the device type is magnetic tape, the routine determines the increment value and stores it in the ICB.
- If the device type is direct access, the module does the following:
  a. It checks for a cylinder change in the ICBSEEK field in the next ICB by comparing it to the cylinder value in the DCBFDAD field in the DCB.
  b. It copies the ICBSEEK field in the next ICB into the DCBFDAD field in the DCB.
  c. If a change in cylinder value was found and the new 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.
- The module attempts to join the channel program for the current buffer to the preceding channel program (that is, chain schedule) by:

  Setting the ICB to not-complete.

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

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

  Updating the SAM IOB prefix block to point to the end of the current channel program.
- It determines whether the joining was successful by testing the ECB (pointed to by the IOB) to see if the I/O supervisor posted the I/O event as completed.
- If the I/O supervisor did not post the event as completed, the joining was successful and the routine returns control to the calling routine.

- If the I/O supervisor did post the event as completed, the routine tests the ICB for the present channel program to find whether the joining was successful or not.
- If the present ICB remains unposted, the present channel program was not joined to the preceding one. The routine prepares to cause restart of the channel by copying the channel program Start address (and the Seek address, if direct-access storage) from the current ICB into the IOB and uses the EXCP macro instruction to cause scheduling of the channel program. It then returns control to the calling routine.
- If the present ICB is posted as completed, the present channel program was joined successfully. (The routine was interrupted long enough, between the joining and the testing, for the channel program to be executed and for the channel-end appendage to post the ICB.) The routine returns control to the calling routine.

Track-Overflow and User-Totaling Save Routines

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 11). Such a block is written by a channel program consisting of a channel program segment for each track to be occupied by a segment of the block. The track-overflow, end-of-block routine computes the address of each track written on; to progress from track to track (to continue writing successive segments of one block), the channel program uses the Search command with the multiple-track (M/T) mode.

The track-overflow and end-of-block modules, IGG019C2 and IGG019T2, are used with output data sets if the access conditions shown in Figure 12 are specified for a DCB. The Open executor selects one of these modules, loads it, and places its address into the DCBEOB or DCBEOBW field. (For an input data set with track-overflow, end-of-block module IGG019CC is used.)

The user-totaling save module is also shown in Figure 12. This module saves an image of the user's totaling area in the sequential access method totaling save area.

User-Totaling Save Module IGG019AX: Module IGG019AX saves an image of the user's totaling area in the sequential access method totaling save area.

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, IGG019TD, 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.

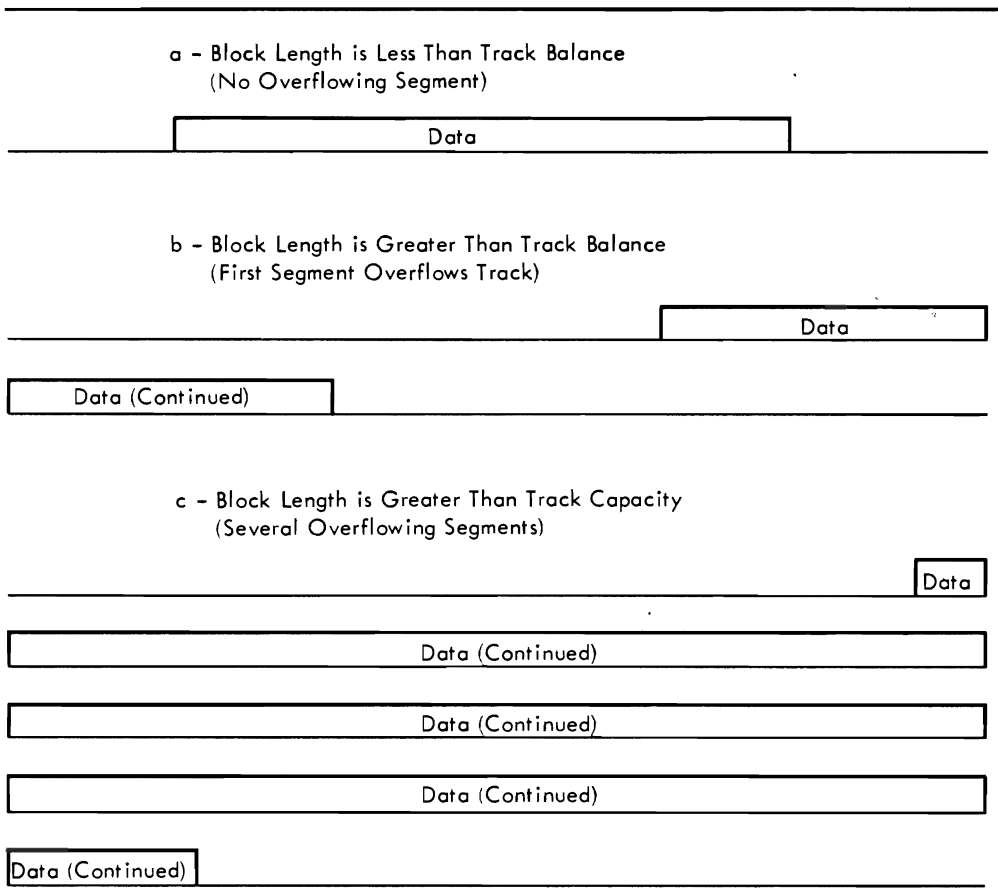


Figure 11. Track-Overflow Records

Access Method Options	Selections					
Track Overflow	X	X	X	X	X	X
Output or,	X	X				
INOUT or OUTIN			X	X	X	X
User Totaling Facility		X		X		X
LABEL=(,,,IN) or LABEL=(,,,OUT) on DD Statement					X	X
End-of-Block Modules						
IGG019AX		AX		AX		AX
IGG019C2	C2		C2		C2	
IGG019T2		T2		T2		T2
IGG019CT[1]					CT	CT

1. This module is described in the previous section, "Ordinary End-of-Block Routines." (See Figure 7)

Figure 12. Module Selector—Track-Overflow, End-of-Block Modules

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

Output, INOUT, or OUTIN

and the DCB specifies:

Track overflow

- If the entire block fits on this track, the module completes a channel program (consisting of one channel program segment) for writing the block, updates the track balance, and passes control to the I/O supervisor. Before the module builds the channel program, tests are made to determine if MSS window processing is needed. If record 1 on track 0 of a cylinder on a page boundary (evenly divisible by 8) is indicated, 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 at least a 1-byte data field fits on this track, the module completes a channel program segment for the segment of the block that fits on the track (by entering the Seek address, storage address, and count field for the channel program segment) and tests if there is another track in the same extent.
- If the next track is in this extent, it compares the remaining block length with the track capacity.
- If the remainder of the block exceeds track capacity, tests are made to determine if MSS window processing is needed. If record 1 on track 0 of a cylinder on a page boundary (evenly divisible by 8) is indicated, 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. The module then proceeds as it does when at least one byte fits on the track.
- If the remainder of the block is less than the track capacity, the module completes the final channel program segment for the final segment of the block, updates the track balance, and passes control to the I/O supervisor.
- If the next track is not in this extent, the module passes control to the track balance routine using an SVC 25 instruction. That routine erases all tracks in the current extent that were found insufficient for the block to be written. On return of control from the track balance routine, the module tests for another extent.
- If there is another allocated extent on this volume, the module reconstructs the channel program 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 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.

End-of-Block Module IGG019T2: 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 user specifies the user-totaling option (that is, if bit 6 in DCBOPTCD is 1) for his data set and 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.
- 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.
- It compares the block length with the space remaining on the track last written on.
- If the entire block fits on this track, the module completes a channel program (consisting of one channel program segment) for writing the block, updates the track balance, and passes control to the I/O supervisor. Before the module builds the channel program, tests are made to determine if MSS window processing is needed. If record 1 on track 0 of a cylinder on a page boundary (evenly divisible by 8) is indicated, 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 at least a 1-byte data field fits on this track, the module completes a channel program segment for the segment of the block that fits on the track (by entering the Seek address, storage address, and count field for the channel program segment) 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.
- If the remainder of the block exceeds track capacity, tests are made to determine if MSS window processing is needed. If record 1 on track 0 of a cylinder on a page boundary (evenly divisible by 8) is indicated, 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. The module then proceeds as it does when at least one byte fits on the track.

- If the remainder of the block is less than the track capacity, the module completes the final channel program segment for the final segment of the block, updates the track balance, and passes control to the I/O supervisor.
- If the next track is not in this extent, the module passes control to the track balance routine using an SVC 25 instruction. That routine erases all tracks in the current extent that were found insufficient for the block to be written. On return of control from the track balance routine, the module tests for another extent.
- If there is another allocated extent on this volume, the module reconstructs the channel program 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 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.

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 13.) 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 14.) These routines:

Are shared by QSAM and BSAM

Only process errors

May be either synchronous or asynchronous

The track-overflow 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 due to 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 as follows:

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

[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 13 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 13 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	X
Put			X			
Input, Readback		X				
Output			X			
Update	X				X	
Paper-tape character conversion				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			
IGG019AT[2]				AT		
IGG019BQ					BQ	

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).

2. This module includes both the paper-tape-synchronizing-and-error-processing routine and the paper tape Get routine. Both routines are described in "Simple Buffering Get Routines" (see Figure 1).

Figure 13. Module Selector—QSAM Synchronizing-and-Error-Processing Modules

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 FEOV (force-end-of-volume) 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 due to an FEOV macro instruction, the module finds the IOBs flagged for output. It then turns off the command-chain flag at the end of the write portion of the channel program, and schedules the write channel programs for execution by means of an EXCP macro instruction.
- When all write channel programs have been executed, 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 due to an FEOV macro instruction, control returns to the routine that passed control.
- If a permanent error is encountered during execution of empty channel programs for an end-of-volume condition or for an FEOV macro instruction, control passes to the SYNAD routine, if one is present. The SYNAD routine returns control to this module.
- The module then processes the error option as follows:

  Accept or Skip option: The pending empty channel programs are rescheduled for execution using an EXCP 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 point to the refill portion of the channel program and issues an EXCP macro instruction for it and all following IOBs.

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

  The module 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 EXCP 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, turns off the commmand-chain flag, and issues an EXCP 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.

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, 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 deciaml 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

non-zero 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 non-zero 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 channel programs. 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 channel program for the next buffer 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 due to an FEOV macro instruction, the module finds the IOBs flagged for output. It then resets the command-chain flag at the end of the empty portion of the channel program and schedules the empty channel programs for execution by means of an EXCP 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 due to an FEOV macro instruction, control returns to the routine that passed control.
- If a permanent error is encountered during execution of empty channel programs for an 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 EXCP macro instructions.

  Terminate: Control passes to the ABEND routine.
- On return of control from the EOV routine module tests whether assembling of a spanned record is in process. If it is being processed, the module turns off the spanned 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 resets the IOB to point to the refill portion of the channel program and issues an EXCP macro instruction for it and all following IOBs.

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

  The module 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 EXCP 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: 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 finds the end of the empty portion of any pending empty-and-refill channel programs, resets the command-chain flag, and issues an EXCP 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 14 for error-processing module selection.

Access Method Options	Selections		
Input, INOUT, OUTIN	X		
Track Overflow	X		
3211 Printer		X	
*, DATA, or SYSOUT specified on DD statement			X
Modules			
IGG019AH			AH
IGG019C1	C1		
IGG019FS		FS	

Figure 14. Module Selector—Error-Processing Modules

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

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

Input, INOUT, or OUTIN

and the DCB specifies:

Track overflow

Get or Read

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

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

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

- It receives control from the supervisor.
- If the segment in error is the last or the only segment of the block, the module posts the ECB with the error code and causes control to be returned to the processing program.
- If the segment in error is not the last segment and is not on an alternate track, the module sets the IOB to address the track following the track in error, posts the ECB with the error code, and causes control to return to the processing program.
- If the segment in error is not the last segment and is on an alternate track, the module increases the track address in the IOB by 1, posts the ECB with the error code, and causes control to be returned to the processing program.

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

- It receives control from the supervisor.
- If the segment in error is the final or only segment of a block amd is not on an alternate track, the module sets the IOB to address the track in error, changes the Read-data command to a NOP command, and issues an EXCP macro instruction for the changed channel program.
- If the segment in error is the final or only segment of a block and is on an alternate track, the module sets the IOB to address the track following the one originally addressed, posts the ECB with the error code, and causes control to be returned to the processing program. (In case of an error in a final or only segment on an alternate track, the remaining blocks on thet track are not read.)
- If the segment in error is not the last one and is not on an alternate track, the module sets the IOB to address the track following the one in error, and issues an EXCP macro instruction for the readdressed channel program.
- If the segment in error is not the last one and is on an alternate track, the module successively increases the track address in the IOB by 1 and issues an EXCP macro instruction for the readdressed channel program.
- When control returns from the I/O supervisor, this module awaits execution of the channel program by using a WAIT macro instruction. On channel program execution, the module restores the purged IOBs (and the Read-Skip command, if it was changed to a NOP command) and causes control to be returned to the processing program.

See Figure 14 for error-processing module selection.

IBM 3211 Printer Asychronous-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 14 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 are 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
- 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.

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 15 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.

End-of-Extent Appendages

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

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

- IGG019AW processes an end-of-extent condition for QSAM update mode channel programs.
- IGG019BM processes an end-of-extent condition for BSAM update mode channel programs.
- IGG019CH processes an end-of-extent condition when neither the update mode nor chained channel-program scheduling is specified.
- IGG019CZ processes end-of-extent conditions when chained channel-program scheduling is used.
- IGG019C4 is loaded in for standard format-F records and verifies whether an extent violation is valid. It is also the end-of-extent appendage for the search-direct option.

Access Method Options[4]	Selections																			
Input, INOUT, OUTIN							X	X	X	X							X	X		
Readback									X											
Update	X	X			X															
Get	X																			
Read		X																		
Offset Read (BDAM)				X																
Create BDAM			X																	
Record format is fixed-length												X								
Record format is fixed-length blocked									X											
Record format is variable-length										X						X				
Record format is variable-length spanned			X	X																
Record format is not fixed-length standard							X	X												
Record format-U						X														
Direct-access storage			X	X			X	X						X						
Printer											X									
Paper tape												X								
Chained Scheduling													X	X						X
Track-overflow															X					
3211 Printer																X				
Magnetic Tape (OPTCD=H)																	X	X		
Search Direct (OPTCD=Z)							X												X	
RPS						X														
V=R																				X
Appendages entered from End-of-Extent Exit																				
IGG019AW	AW																			
IGG019BM		BM																		
IGG019CH								CH												
IGG019CZ														CZ						
IGG019C4							C4													
Appendages entered from SIO Exit																				
IGG019CG					CG															
IGG019CL											CL									
Appendages entered from Channel-End Exit																				
IGG019BT			BT																	
IGG019BV				BV																
IGG019CI									CI											
IGG019CJ										CJ										
IGG019CS												CS								
IGG019CU													CU							
IGG019C0						C0														
IGG019C3[2]					C3															
IGG019EI																		EI		
IGG019EJ																	EJ			
IGG019FP																			FP	
IGG019V6[1]																				V6
Appendage entered from PCI Exit																				
IGG019V6																				V6
Appendages entered from Abnormal End Exit																				
IGG019CU[1]													CU							
IGG019CI[3]									CI											
IGG019CJ[3]										CJ										
IGG019C0[3]						C0														
IGG019C3															C3					
IGG019EI																		EI		
IGG019EJ																	EJ			
IGG019FR																FR				
IGG019V6[1]																				V6

1. Module has multiple entry points. Description appears in Program Controlled Interruption Appendages.
2. Module has multiple entry points. Description appears in Abnormal-End Appendages.
3. Module has multiple entry points. Description appears in Channel-End Appendages.
4. If *, DATA, or SYSOUT are specified on the DD statement, no appendages are loaded.

Figure 15. Module Selector—Appendages

Appendage IGG019AW (End-of-Extent—Update—QSAM): Appendage IGG019AW readdresses the refill portions of all QSAM update channel programs to a new extent. The Open executor selects and loads this module for use as the end-of-extent appendage if the Open parameter list specifies:

Update

and the DCB specifies:

Get

The appendage operates as follows:

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

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

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

- It tests to see if the EXCP was issued from an SVC routine. If so, it effects a normal return to EXCP.
- If the interruption occurred in a Seek CCW, the appendage copies the Seek address from the refill portion of the present channel program into the DCB and proceeds to check to determine if there is another extent.
- If there is no other extent, the appendage sets error indications in the IOB and returns control to the EXCP supervisor. The EXCP supervisor then issues a PURGE macro instruction for that channel program. The update synchronizing routine ensures writing out of the empty portions of pending channel programs.
- If the interruption occurred in a Read-count CCW and there is a new extent, the appendage builds a Seek address for the new extent using the starting address from the DEB. It then copies this new Seek address into the DCB and IOSEEKA and updates the M value in the refill portion of each channel program.
- When using the rotational position sensing (RPS) feature and when the Seek address is updated to reflect the beginning of the next extent, the Set-sector byte is reset to 0.
- It resets the IOB to address the next track and its channel program and returns control to the I/O supervisor.

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

Update

and the DCB specifies:

Read

The appendage operates as follows:

- It receives control from the EXCP supervisor when a channel program to refill a buffer attempts to read the first block beyond the present extent.
- It tests to see if the EXCP was issued from an SVC routine. If so, it effects a normal return to EXCP.
- If there is no other extent for a Refill channel program, the appendage sets error indications in the IOB and returns control to the EXCP supervisor.
- If there is a new extent for a Refill channel program, the appendage adds 1 to the value of M in the DCBFDAD field and in the Seek address of each refill channel program for the DCB. It places the new Seek address into the current IOB and returns control to the EXCP supervisor. The supervisor restarts the channel program.
- When the Seek address is updated to reflect the beginning of the next extent and the rotational position sensing (RPS) feature has been specified, the Set-sector byte is reset to zero.

Appendage IGG019CH (End-of-Extent—Ordinary): Appendage IGG019CH finds a new extent when the EXCP supervisor finds an end-of-extent condition. The Open executor selects and loads this appendage for use as the end-of-extent appendage if the Open parameter list specifies:

Input, INOUT, or OUTIN

and the DCB specifies:

Direct-access storage device

Record format other than fixed-length standard

Normal channel-program scheduling

The appendage operates as follows:

- It receives control when a channel program attempts to read a block beyond the present extent.
- It tests to see if the EXCP was issued from an SVC routine. If so, it effects a normal return to EXCP.
- The appendage examines the DEB for another extent.
- If there is another extent, the appendage enters the new full device address in the DCB, IOSEEKA, the IOBs, and returns control to the EXCP supervisor. The EXCP supervisor restarts the channel program.
- When the SEEK address is updated to reflect the beginning of the next extent and RPS was specified, the set-sector byte is set to zero.
- If there is no other extent, the appendage sets error indications in the IOB and the DCB to show an end-of-volume condition and returns control to the EXCP supervisor. The EXCP supervisor then issues a PURGE macro instruction for that channel program.

Appendage IGG019CZ (End-of-Extent—Chained Channel-Program Scheduling): Appendage IGG019CZ readdresses the chain of channel programs to a new extent when the EXCP supervisor finds an end-of-extent condition. The Open executor selects and loads this appendage for use as the end-of-extent appendage if the DCB specifies:

Chained channel-program scheduling

Direct-access storage device

The appendage operates as follows:

- It tests to see if the EXCP was issued from an SVC routine. If so, it effects a normal return to EXCP.
- It receives control when an end-of-track condition interrupts the chained scheduling and the I/O supervisor finds that the next track is not in the current extent.
- If there is another extent, the appendage enters the new Seek address in the DCB, IOB,IOSEEKA, updates the Seek addresses of the remaining ICBs, resets the error indications in the first ICB, resets the sector value to zero when RPS is specified, and returns control to the I/O supervisor to reschedule the channel program for execution.
- If there is no other extent, the appendage sets a volume-full indication in the DCB, IOB, and ICB and returns control to the I/O supervisor to skip further scheduling for this DCB.

Appendage IGG019C4 (End-of-Extent for Search Direct): Appendage IGG019C4 finds a new extent when the EXCP supervisor finds an end-of-extent condition. The Open executor selects and loads this appendage for use as the end-of-extent appendage if the Open parameter list specifies:

Input, INOUT, or OUTIN

and the DCB specifies:

Direct-access storage device

Record format other than fixed-length standard

Normal channel-program scheduling

Search direct

The appendage operates as follows:

- It receives control when a channel program attempts to read a block beyond the present extent.
- It tests to see if the EXCP was issued from an SVC routine. If so, it effects a normal return to EXCP.
- If another extent is not available and all of the following conditions exist:

  the Seek is not on cylinder FFnn and

  M/T Read-count command caused the Seek beyond the current extent and

  unit check is on

IGG019C4 does the following:

clears all error and status flags in the IOB,

sets CHAN and DEV END in the IOBCSW,

sets 7F in the IOB completion code,

clears the DCB flags, and

sets FF in the HI cylinder byte of the next IOBSEEK field (multiple IOBs) or the DCBFDAD field (one IOB).

It then performs normal channel-end processing (because the channel-end appendage is bypassed) and sets register 14 code to "skip" (because the required I/O has been completed).

- If any of the following conditions exist:

  M/T Read-count did not cause the Seek or

  cylinder FFnn is found or

  unit check is off

  the appendage does the following:

  sets the volume-full list in the DCBCIND1 field,

  sets the unit-exception bit in the CSW,

  sets the register 14 code to "extent error," which is interpreted as an end of file condition, and returns.

- If there is another extent, the appendage enters the new full disk address in the DCB, IOSEEKA, and the IOBs. It resets the sector value to zero when RPS is specified.
- For seeks on record 0, it changes the second TIC in the search direct channel program to an M/T Read Count to make it a search previous channel program. It then sets the register 14 code to "try again," and returns.

Start I/O (SIO) Appendages

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

- IGG019CG. This appendage makes the Seek address accessible to the I/O supervisor for QSAM and BSAM update channel programs that refill buffers. (This is necessary because the Seek address for such a channel program is read by the preceding channel program into a location unknown to the I/O supervisor.)
- IGG019CL. This appendage causes the next line to print at the top of a new page if a printer overflow condition was encountered in the execution of the last channel program.

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.

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

Update

The appendage operates as follows:

- It tests to see if the EXCP was issued from an SVC routine. If so, it effects a normal return to EXCP.
- It tests the IOB to determine whether the buffer is to be emptied and refilled or to be refilled only.
- It performs new extent checking and moves the new extent address into IOSEEKA.
- With the rotational position sensing (RPS) feature, the offset to the special FDAD from the Read CCW is not the same for record-ready. A test is made for record-ready and the correct offset used. When the special FDAD has been partially or completely destroyed, the channel program Start address is set to point to a TIC so the IOB will ultimately be marked in error. The offset of the TIC is different for record-ready.

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.

Channel-End Appendages

Channel-end appendages, if present, gain CPU 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.

When the rotational position sensing (RPS) function is implemented in the

channel programs, the Read-sector follows the Read-data command and wipes out the Residual count for the Read. The implementation of RPS requires reading in the full 8 bytes of the count field of the record to use the DLDL plus the K of the count for checking the number of bytes read. The count is read into an 8-byte area following the channel program. In the channel-end appendages, it is necessary to move the CCHHR into the next search argument or DCBFDAD+3 depending on the number of IOBs.

The channel-end appendages are:

- IGG019BT. This appendage schedules the writing of successive blocks when a record has to be segmented.
- IGG019BV. This appendage distinguishes between valid wrong-length blocks and variable-length blocks.
- 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.
- IGG019CS. This appendage distinguishes between valid and invalid wrong-length indications when paper tape is being read.
- IGG019C0. This appendage distinguishes between wrong-length and undefined-length blocks when format-U records are being read from RPS direct-access devices.
- 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.
- IGG019FP. This appendage does length checking for all formats supported by the search-direct feature.

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: 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.

Channel-End Appendage IGG019BV (Offset Read): Appendage IGG019BV distinguishes between valid wrong-length blocks and variable-length blocks. It also performs an offset read function when necessitated by spanned records. The Open executor selects and loads this appendage and the associated Read module (IGG019BU), 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

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

The appendage operates as follows:

- It receives control from the I/O supervisor at the channel-end exit.
- If the appendage finds a unit exception bit on in the CSW, it returns to the I/O supervisor immediately.
- If the unit check bit is on, the Abnormal routine is branched too. The abnormal channel-end appendage returns to the I/O supervisor immediately if it finds a cylinder-end or file-protect condition. Otherwise, the current channel program is changed back to Read-key-and-data, and control is returned to the I/O supervisor.
- If a key was not read (Read-data CCW), the command is changed back to Read-key and data.
- If a key was expected (Read-key-and-data CCW) and there was no key to read (key length=0 in count just read), then the Read CCW must be rescheduled with an offset.
- The appendage calculates the length of the block and compares it to the block length field.
- If the lengths are equal, it resets error indicators in the ECB.
- If the lengths are unequal and the current channel program is changed to Read-key-and-data, control is returned to the I/O supervisor. The I/O supervisor then sets the ECB to show that the channel program executed with an error condition.
- The appendage checks the SDW to see if another segment is to follow. If there is, the next channel program is changed to Read-data.

Appendage IGG019CI (Channel End, Abnormal 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, OUTIN, or UPDAT

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.
- If the IOBUPERR flag is off (abnormal-end not entered before for this I/O request), it transfers control to the common channel-end routine.
- If the IOBUPERR flag is on (abnormal-end entered before for this I/O request), the channel-end appendage does the following:

  Turns off the IOBUPERR flag.

  Turns back on the command chain for the last CCW of the current segment, which is the last CCW executed.

  Changes the IOBSEEK argument to the same seek argument pointed to by the seek CCW that follows the current segment.

  Changes the IOBSTART address so that it points to the CCW after the seek CCW that follows the current segment. The CCW pointed to is either set sector or search ID.
- The channel program is then restarted from that point.

The abnormal-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.
- If the error is not yet a permanent error, the abnormal-end appendage does the following:

  If the IOBUPERR flag is on (prior abnormal-end entry into this section), it gives control to the ERPs, via EXCP, for retry.

  If the IOBUPERR flag is off, it branches to the scan routine to get the address of the last CCW in the current segment.

  If the last CCW in the segment is the last CCW in the channel program and IOBSTART pointed to the beginning of the segment on entry, the appendage gives control to the ERPs, via EXCP, for retry.

  If the last CCW in the segment is not the last CCW in the channel program, the appendage turns off the command chain on the last CCW. The IOBSTART and IOBSEEK have been updated to the current segment. The appendage turns on the IOBUPERR flag and returns control to EXCP for restart.
- If the error is a permanent error, the abnormal-end appendage does the following:

  If the IOBUPERR flag is off, it transfers control to the common channel-end routine.

  If the IOBUPERR flag is on, it turns on the command chain of the failing segment, turns off the IOBUPERR flag, and transfers control to the common channel-end routine.

The scan routine operates as follows:

- It establishes the address of the last CCW in the current segment and passes it back to the calling routine.
- It assumes that the current segment begins with the CCW pointed to by IOBSTART.

- It searches from the beginning of the segment until it finds a seek, NOP, or CCW without a command chain (other than TIC).
- It passes the end-of-segment address to the caller. The address is either the first CCW without a command chain or the seek address minus eight, whichever comes first.

The common channel-end routine operates as follows:

- It performs length checking for fixed-length records. If the record format is fixed standard or the track-overflow feature is used with record-ready, the SLI bit is left off in the Read-data CCW. 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 and the sector value for this last record is used for the next READ.
- Length checking for nonstandard format-F records without the track-overflow feature is performed in the following manner.

The module searches the channel program for a Read-count command, picks up the address of the count to locate the data length and key length, and adds them together. The appendage then compares this value to the block size to determine whether a short block was read. (The SLI bit is on so the channel program will not be terminated with the Read-data CCW.) If a short block has been read, the appendage divides the data length plus key length by the LRECL to determine whether the record is a multiple of the LRECL. If it is, the appendage continues processing using code common to non-RPS. If the DD is not a multiple of the LRECL, the incorrect length bit in the CSW is turned on and processing continues with code common to both RPS and non-RPS.

- For record-ready channel programs, if there is only one IOB, the CCHHR of the count is moved into the DCBFDAD3. If there is more than one IOB, the CCHHR of the count is moved into the IOBSEEK+3 of the next IOB or, in the case of update, into the next special FDAD+3.

Appendage IGG019CJ (Channel End, Abnormal End—Variable-Length Record Format): Appendage IGG019CJ distinguishes between valid wrong-length blocks and variable-length blocks. The Open executor selects and loads this appendage if the Open parameter list specifies:

Input, INOUT, OUTIN, or UPDAT

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. When the track-overflow option is used with rotational position sensing (RPS), no length checking is performed because the count that would be read in is the count of the first segment only. When the record-ready feature is used without track-overflow, all 8 bytes of the count of the record are read into virtual storage. The DLDL plus the K of the count is compared with the LL of the record. If they are equal, the module branches via the return register. If

they are not equal, it turns on the wrong-length indicator, dummies up the residual count, and continues processing with code common to both RPS and non-RPS.

For record-ready, if there is only one IOB, the CCHHR of the count is moved into the DCBFDAD+3. If there is more than one IOB, the CCHHR of the count is moved into the IOBSEEK+3 of the next IOB or, in the case of update, into the next special FDAD+3.

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 IOBUPERR flag is off (abnormal-end not entered before for this I/O request), it transfers control to the common channel-end routine.
- If the IOBUPERR flag is on (abnormal-end entered before for this I/O request), the channel-end appendage does the following:

  Turns off the IOBUPERR flag.

  Turns back on the command chain for the last CCW of the current segment, which is the last CCW executed.

  Changes the IOBSEEK argument to the same seek argument pointed to by the seek CCW that follows the current segment.

  Changes the IOBSTART address so that it points to the CCW after the seek CCW that follows the current segment. The CCW pointed to is either set sector or search ID.
- The channel program is then restarted from that point.

The abnormal-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.
- If the error is not yet a permanent error, the abnormal-end appendage does the following:

  If the IOBUPERR flag is on (prior abnormal-end entry into this section), it gives control to the ERPs, via EXCP, for retry.

  If the IOBUPERR flag is off, it branches to the scan routine to get the address of the last CCW in the current segment.

  If the last CCW in the segment is the last CCW in the channel program and IOBSTART pointed to the beginning of the segment on entry, the appendage gives control to the ERPs, via EXCP, for retry.

  If the last CCW in the segment is not the last CCW in the channel program, the appendage turns off the command chain on the last CCW. The IOBSTART and IOBSEEK have been updated to the current segment. The appendage turns on the IOBUPERR flag and returns control to EXCP for restart.

- If the error is a permanent error, the abnormal-end appendage does the following:

  If the IOBUPERR flag is off, it transfers control to the common channel-end routine.

  If the IOBUPERR flag is on, it turns on the command chain of the failing segment, turns off the IOBUPERR flag, and transfers control to the common channel-end routine.

The scan routine operates as follows:

- It establishes the address of the last CCW in the current segment and passes it back to the calling routine.
- It assumes that the current segment begins with the CCW pointed to by IOBSTART.
- It searches from the beginning of the segment until it finds a seek, NOP, or CCW without a command chain (other than TIC).
- It passes the end-of-segment address to the caller. The address is either the first CCW without a command chain or the seek address minus eight, whichever comes first.

The common channel-end routine operates as follows:

- 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 blocksize, 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 IGG019CS (Channel End—Paper Tape): Appendage IGG019CS distinguishes between valid wrong-length blocks and the wrong-length indication characteristic when paper tape is being read. The Open executor selects and loads this appendage if the DCB specifies:

Fixed-length record format

Paper tape

The appendage operates as follows:

- It receives control when the I/O interruption supervisor arrives at the channel-end exit.
- It tests to see if the EXCP was issued from an SVC routine. If so, it effects a normal return to EXCP.
- If the channel status word (IOSCSW) residual count is zero, the appendage turns off error indications in the IOB and then returns control to the I/O supervisor.

- If the channel status word (IOSCSW) residual count is not zero, the appendage returns control to the I/O supervisor immediately.

Appendage IGG019C0 (Channel End, Abnormal End—Format-U and RPS Direct Access): This appendage updates the residual count field of the IOB CSW with the length of a variable-length block when format-U records are being read from RPS devices. The Open executor selects and loads this appendage if the Open parameter list specifies:

Input, Output, Update, INOUT, or OUTIN

and the DCB specifies:

Undefined-length records

no track-overflow

and the device is a direct-access with RPS.

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.
- If the IOBUPERR flag is off (abnormal-end not entered before for this I/O request), it transfers control to the common channel-end routine.
- If the IOBUPERR flag is on (abnormal-end entered before for this I/O request), the channel-end appendage does the following:

  Turns off the IOBUPERR flag.

  Turns back on the command chain for the last CCW of the current segment, which is the last CCW executed.

  Changes the IOBSEEK argument to the same seek argument pointed to by the seek CCW that follows the current segment.

  Changes the IOBSTART address so that it points to the CCW after the seek CCW that follows the current segment. The CCW pointed to is either set sector or search ID.

  The channel program is then restarted from that point.

The abnormal-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.
- If the error is not yet a permanent error, the abnormal-end appendage does the following:

  If the IOBUPERR flag is on (prior abnormal-end entry into this section), it gives control to the ERPs, via EXCP, for retry.

  If the IOBUPERR flag is off, it branches to the scan routine to get the address of the last CCW in the current segment.

  If the last CCW in the segment is the last CCW in the channel program and IOBSTART pointed to the beginning of the segment on entry, the appendage gives control to the ERPs, via EXCP, for retry.

  If the last CCW in the segment is not the last CCW in the channel program, the appendage turns off the command chain on the last CCW. The IOBSTART and IOBSEEK have been updated to the current

segment. The appendage turns on the IOBUPERR flag and returns control to EXCP for restart.

- If the error is a permanent error, the abnormal-end appendage does the following:

  If the IOBUPERR flag is off, it transfers control to the common channel-end routine.

  If the IOBUPERR flag is on, it turns on the command chain of the failing segment, turns off the IOBUPERR flag, and transfers control to the common channel-end routine.

The scan routine operates as follows:

- It establishes the address of the last CCW in the current segment and passes it back to the calling routine.
- It assumes that the current segment begins with the CCW pointed to by IOBSTART.
- It searches from the beginning of the segment until it finds a seek, NOP, or CCW without a command chain (other than TIC).
- It passes the end-of-segment address to the caller. The address is either the first CCW without a command chain or the seek address minus eight, whichever comes first.

The common channel-end routine operates as follows:

- If processing a partitioned data set, it returns immediately to the I/O supervisor.
- It returns to the I/O supervisor if the operation was either write or backspace.
- Because of the implementation of RPS function in the channel programs, the Read-sector follows the Read-data command and wipes out the residual count for the Read. The implementation of RPS requires reading the full 8-byte count field of the record to use the DLDL plus the K of the count for checking the number of bytes read. The count field is read into a 8-byte area following the channel program. This appendage moves the CCHHR into the next search argument or DCBFDAD+3 depending on the number of IOBs. In addition, this routine adds DLDL with K and stores the result in the Residual Status Word of the IOB.

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 is comprised of 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 two 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 two bytes of the checkpoint header record do not satisfy the length check as a variable-length record. If the first two 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 two 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 decremented 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 decremented, the IOB flag fields re-initialized, 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 that 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 is comprised of 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 IGG019FP (Channel End—Search-Direct): Appendage IGG019FP receives control at channel-end time or if an incorrect length has been given.

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 does length checking for all formats supported by the search-direct feature.
- If the second TIC in the channel program is a multitrack Read-count CCW, the appendage moves the 2-byte data length of the count field, pointed to by the address of the multitrack Read-count CCW, to the right half of the second TIC location.
- For format-V records, it compares the data length to the record descriptor word that is pointed to by the Read-data command.
- For format-U records, no length checking is provided.
- It finds the multitrack Read-count CCW following the Read-data and moves the data length of the count field, pointed to by the Read-count CCW, to the right half of the second TIC of the next IOB (or to that of the same IOB, if only one IOB exists).
- It moves the CCHHR portion of the count, pointed to by the multitrack Read-count CCW following the Read-data CCW, to the next IOBSEEK field (for multiple IOBs) or to the DCBFDAD (for one IOB).
- It changes the multitrack Read-count CCW, preceding the Read-data CCW to a TIC CCW to the Read-data CCW.

- For exchange buffering, the value in the DCBBLKSI field is used instead of the data length specified in the Read-data command.
- It changes the second TIC in the search direct channel program to a M/T Read Count to make it a search previous channel program.

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 CPU control from the PCI appendage.

This appendage performs the following three functions:

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

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

Appendage IGG019CU (PCI, 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 checks the channel program for a rotational position sensing (RPS) program and, if one is found, moves the ICB's channel program address to the main IOB's TIC, which has been offset by the Set Sector CCW.
- 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 error.

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

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

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

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

  It repeats this 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 returns control to the I/O interruption supervisor. That routine then processes the channel program in the error-retry procedure.

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

- It receives control after the I/O supervisor error-retry procedure is found unsuccessful in correcting the error.
- 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 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 since it does not hold the local lock.

Abnormal-End Appendages

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

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

Appendage IGG019C3 (Channel End and Abnormal End—Track Overflow, QSAM Update): Appendage IGG019C3 schedules the track-overflow-asynchronous-error-processing routine if a permanent error occurs in a channel program for an input data set with track overflow. The Open executor selects and loads this appendage for use as the abnormal-end appendage if the Open parameter list specifies:

Input, INOUT, OUTIN, or UPDAT

and the DCB specifies:

Track overflow

or if the Open parameter list specifies:

UPDAT

and the DCB specifies:

QSAM

The channel-end appendage operates as follows:

- It receives control from the I/O interruption supervisor when the latter reaches the channel end or 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.
- If the IOBUPERR flag is off (abnormal-end not entered before for this I/O request), it transfers control to the common channel-end routine.
- If the IOBUPERR flag is on (abnormal-end entered before for this I/O request), the channel-end appendage does the following:

  Turns off the IOBUPERR flag.

  Turns back on the command chain for the last CCW of the current segment, which is the last CCW executed.

  Changes the IOBSEEK argument to the same seek argument pointed to by the seek CCW that follows the current segment.

  Changes the IOBSTART address so that it points to the CCW after the seek CCW that follows the current segment. The CCW pointed to is either set sector or search ID.

  The channel program is then restarted from that point.

The abnormal-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.

- If the error is not yet a permanent error, the abnormal-end appendage does the following:

  If the IOBUPERR flag is on (prior abnormal-end entry into this section), it gives control to the ERPs, via EXCP, for retry.

  If the IOBUPERR flag is off, it branches to the scan routine to get the address of the last CCW in the current segment.

  If the last CCW in the segment is the last CCW in the channel program and IOBSTART pointed to the beginning of the segment on entry, the appendage gives control to the ERPs, via EXCP, for retry.

  If the last CCW in the segment is not the last CCW in the channel program, the appendage turns off the command chain on the last CCW. The IOBSTART and IOBSEEK have been updated to the current segment. The appendage turns on the IOBUPERR flag and returns control to EXCP for restart.

- If the error is a permanent error, the abnormal-end appendage does the following:

  If the IOBUPERR flag is off, it transfers control to the common channel-end routine.

  If the IOBUPERR flag is on, it turns on the command chain of the failing segment, turns off the IOBUPERR flag, and transfers control to the common channel-end routine.

The scan routine operates as follows:

- It establishes the address of the last CCW in the current segment and passes it back to the calling routine.
- It assumes that the current segment begins with the CCW pointed to by IOBSTART.
- It searches from the beginning of the segment until it finds a seek, NOP, or CCW without a command chain (other than TIC).
- It passes the end-of-segment address to the caller. The address is either the first CCW without a command chain or the seek address minus eight, whichever comes first.

The common channel-end routine operates as follows:

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

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.

The appendage operates as follows:

- The module receives control before and after the error recovery procedure (ERP).
- It tests to see if the EXCP was issued from an SVC routine. If so, it effects a normal return to EXCP.
- The first time the abnormal-end appendage is entered, it is returned to the I/O supervisor to schedule the 3211 ERP.
- The second time the appendage is entered, a return to the I/O supervisor is made when any of the following occurs:

  Command retry bit in sense byte 1 is off.

  Error persists after the print line operation was retried.

  Otherwise, the abnormal-end appendage obtains the address of the interruption request block (IRB) from the DEB and the address of the interruption queue element (IQE) from the IRB. The IQE address is placed in register 1 in complement form. The address of the stage 2 exit effector is obtained from the communications vector table (CVT) and a branch is taken to that address. The stage 2 exit effector schedules the asynchronous routine, which retries the print line. It is then returned to the I/O supervisor.

QSAM Control Routines

These control routines, shared by QSAM and BSAM, consist of both modules loaded by the Open executor and macro expansions. The selection and loading of one of the modules is 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 16 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.

There are two PRTOV routines, which are macro expansions. Whenever the assembler encounters either of the two macro instructions shown in Figure 17, it substitutes the corresponding macro expansion in the processing program object module.

Access Method Options	Selections		
CNTRL	X	X	X
Printer	X		
Card Reader, Single Buffer		X	
3525 Printer			X
Modules			
IGG019CA		CA	
IGG019CB	CB		
IGG019FA			FA

Figure 16. Module Selector—Control Modules

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

Figure 17. 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 instructions are encountered in a processing program. The track balance routine receives control from a Write routine or the track-overflow, end-of-block routine.

The BSAM control routines are described later in this section of the manual. See Figures 20, 21, and 22 for information about control modules.

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 18 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.

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

Access Method Options	Selections										
Input	X		X	X		X					X
Output		X			X			X	X	X	X
INOUT, OUTIN	X	X									X
Update							X				
Read	X		X	X			X				X
Offset Read						X					
Write		X									X
Write (Load mode) (Create-BDAM)					X			X	X	X	
Paper-tape character-conversion			X	X							
Fixed-length record format			X					X		X	X
Undefined-length record format				X					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											
IGG019BA	BA	BA									
IGG019BF			BF	BF							
IGG019BH							BH				
IGG019BR					BR						
IGG019BU						BU					
IGG019DA								DA			
IGG019DB									DB		
IGG019DD										DD	
IGG019DK											DK

Figure 18. Module Selector—Read and Write Modules

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 completes the channel program by inserting the buffer address from the DECB, and the length from either the DECB (for undefined-length records), the DCB (for fixed-length records, and for input of variable-length records), or the record itself (for output of variable-length records).

- If a block is to be written on a direct-access storage device, the module tests the DCBOFLGS field in the DCB to establish the validity of the value in the DCBTRBAL field.
- If the DCBTRBAL value is valid, or if a block is to be written on a device other than direct-access storage, or if a block is to be read from any device, the module passes control to an end-of-block routine.
- If the DCBTRBAL value is not valid (that is, the preceding operation was a Read, Point, or Open for MOD), the module issues an SVC 25 instruction to pass control to BSAM control module IGC0002E to obtain a valid track balance. When control returns to this module, it passes control to an end-of-block routine.

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

Read

Fixed-length or undefined-length record format

Paper tape

The module operates as follows:

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

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

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

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

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

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

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

The tables used for code conversion are listed along with the code conversion routines in Appendix A.

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

With the rotational position sensing (RPS) feature, this module bypasses the new CCWs when necessary.

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, sets it to point to the Read channel program, and copies the length and buffer address from the DECB or the DCB into the Read CCW.
- If data is to be written from a buffer, the module flags the IOB for a Write operation, sets it to point to the Write channel program, copies the auxiliary storage address from the DCBFDAD field into the IOBSEEK field, and completes the length and buffer address entries in the Write CCW.
- The module passes control to end-of-block module IGG019CC. On return of control from that module, it returns control to the processing program.

Write Module 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 as 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 completes the channel program by inserting the buffer address and the record length. The buffer address is obtained for the DECB. If there is no key with this segment (this is not the first segment), the buffer address is offset by the key length. This determination is made by checking to see if the CCW has been changed by module IGG019BV to Read-data. The record length is obtained from the DEB and modified by the key length if appropriate.
- The module then issues an EXCP 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, IGG019BB, processes the CHECK macro instruction for SYSIN (see Figure 19). 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 blocksize 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 *OS/VS2 Data Areas,* SYB8-0606.
- 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.
- A JES PUT request is issued. The request is issued as many times as is 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 form converted from decimal to binary. For format-D records when BUFOFF $\neq$ F, the length of the record read is calculated and placed in the DCB LRECL field.

Figure 19 lists the available Check routines and the conditions that cause a particular module to be used. The Open executor selects one of the four routines, loads it, and places its address into the DCBCHECK field. For example, Figure 19 shows that module IGG019BG is selected and loaded if Read and paper-tape character conversion are specified.

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

Access Method Options	Selections								
Input		X				X			
* or DATA specified on DD statement						X			
Output	X			X			X		
SYSOUT specified on DD statement							X		
INOUT, OUTIN	X								
Update			X						
Read		X							
Write	X								
Write (Load) (Create-BDAM)				X	X				
Paper-tape character-conversion		X							
Variable-length spanned record format				X					
Check Modules									
IGG019BB	BB	BB				BB			
IGG019BG		BG							
IGG019BI			BI						
IGG019BS				BS					
IGG019DC					DC				
IGG019DK[1]							DK		

1. The Check routine described in this section is part of the BSAM processing module IGG019DK listed in Figure 18.

Figure 19. Module Selector—Check Modules

entire input buffer is translated from ASCII to EBCDIC. 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 14.) 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 IGG019BG (Paper-Tape Character-Conversion): Module IGG019BG processes error conditions detected by Read module IGG019BF.

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

The module operates as follows:

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

Check Module IGG019BI (Update): Module IGG019BI synchronizes the execution of a BSAM update channel program to the progress of the processing program. A BSAM update channel program either writes data from a buffer or reads data into a buffer.

The module also 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 14) 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 the "SAM/PAM SVC Routines" section.
- 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 20 and Figure 21 list the various kinds of control routines and the conditions that cause them to gain control. Figure 20 shows the access condition options that cause the Open executor to load a control routine for use with a DCB.

Figure 21 lists the different macro expansions constructed by the assembler.

Control Module IGG019BC (Note/Point—Direct Access): The Open executor selects and loads this module if the DCB specifies:

Point

Direct-access storage device

The module consists of two routines: Note and Point.

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

If the records are standard format without the track-overflow feature, the record number is passed to the resident sector routine to compute the sector value. If the record format is not standard F or if the track-overflow feature is used, the value X'FF' is placed in the byte used by the Set-sector CCW.

The Note routine operates as follows:

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

Access Method Options	Selections								
Note/Point	X	X		X	X				
Update, Track Overflow				X					
Chained Scheduling				X	X				
CNTRL			X			X	X	X	
Direct-Access Storage	X			X					
Magnetic Tape		X	X		X				
Card Reader						X			
Printer							X		
3525 Printer								X	
Control Modules									
IGG019BC	BC								
IGG019BD		BD							
IGG019BE			BE						
IGG019BK				BK					
IGG019BL					BL				
IGG019CA[1]						CA			
IGG019CB[1]				CB					
IGG019FA[1]					FA				

1. These routines are also used in QSAM; see Figure 16 for description of these routines.

Figure 20. Module Selector—Control Modules Selected and Loaded by the Open Executor

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

1. These routines are also used in QSAM; see the QSAM section for a description of the routines.
2. This table duplicates Figure 17; it is repeated here to identify all control routines available in BSAM.

Figure 21. Control Routines that Are Expansions of Macro Instructions [1,2]

- It obtains the FDAD value used by the channel program last executed. The address is found in either the IOB or the DCB depending 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 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 IGG019BC converts a relative address (of the form TTRZ) to the full direct-access device address (FDAD) used by the next channel program to read or write the block noted.

The Point routine operates as follows:

- It receives control when a POINT macro instruction is encountered in a processing program.
- It issues a BALR instruction to pass control to the IECPCNVT routine 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 routine sets the DCBIFLGS and IOBECBCC fields to show that an addressing error occurred before returning control. (The Check routine finds the error and processes accordingly.)
- It establishes the actual value to be used by the next channel program by testing the fourth byte of the relative address TTRZ. If the value of Z is zero, the full address is decreased by one; if Z is one, the address calculated by the IECPCNVT routine is left unchanged. For an explanation of how the value of Z is set, refer to the description of the POINT macro instruction in *OS/VS Data Management Macro Instructions.*
- It inserts the value in the DCBFDAD and IOBSEEK fields, sets the DCBOFLGS field to show that the contents of the DCBTRBAL field are no longer valid, and returns control to the processing program.

Control Module IGG019BD (Note/Point—Magnetic Tape): 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 the special access conditions of chained scheduling, track overflow, and update. The Open executor selects and loads this module if the DCB specifies:

Point

Direct-access storage

Chained scheduling, track overflow, or the Open parameter is update.

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

If the records are standard F without the track-overflow feature, the record number is passed to the resident sector routine to compute the sector value. If the record format is not standard F or if the track-overflow feature is used, the value 255 is placed in the byte used by the Set-sector CCW.

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 track-overflow or update is being used, and into the ICBSEEK field if chained scheduling is used.
- 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.

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 one; if Z is one, the address calculated by the convert routine is left unchanged. For an explanation of how the value of Z is set, refer to the description of the POINT macro instruction in *OS/VS Data Management Macro Instructions.*
- It inserts the value into the DCBFDAD and IOBSEEK fields if track overflow or update is being used, and also into the ICBSEEK field if chained scheduling is used. It sets the DCBOFLGS field to show that the contents of the DCBTRBAL field are no longer valid and returns control to the processing program.

Control Module IGG019BL (Note/Point—Magnetic Tape—Chained Scheduling): Module IGG019BL is selected and loaded by the Open executor if the DCB specifies:

Point

Magnetic tape

Chained scheduling

The module consists of two routines: Note and Point.

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

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

- It receives control when a POINT macro instruction is encountered in a processing program.
- A channel program is constructed to read forward or backward one block.
- The channel program is passed for execution the number of times required to position the tape at the desired block.
- The last spacing channel program is followed by a NOP channel program to obtain device-end information for the last spacing operation.

- Control is returned to the processing program, unless a tapemark, load point, or permanent error is encountered in the execution of one of the channel programs. In that case, the routine sets the DCBOFLGS field to indicate a permanent error before returning control to the processing program. (Subsequent attempts by the Read or Write routine to cause scheduling of channel programs for execution results in their not being scheduled. On the next entry into the Check routine, the condition is detected and handled.)

Basic Partitioned Access Method 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. (Refer to the BSAM portion of this publication.) A processing program using BPAM routines for input from the directory is presented with the address of a member in a channel program or in a table; 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

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 22) differ from BSAM and QSAM routines in that BPAM routines are not loaded at Open time; the Stow routine is loaded at execution time, all the coding for Find (C option) is a macro expansion, and the Find (D option)/BLDL routine and the converting routines are in virtual storage. Figure 22 shows how these routines gain control.

See the "SVC Routines" section 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 Area	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 22. 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 refer to *OS/VS2 Open/Close/EOV Logic*. Figure 23 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 24, 25, & 26	See Diagram E	XCTL (WTG Table)	See Diagram E
CLOSE	See Figure 27	Close Routine	XCTL (WTG Table)	Close Routine See Figure 27

Figure 23. 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 his 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, refer to *OS/VS2 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 *OS/VS2 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 *OS/VS2 System Programming Library: Data Management.* The logic of the DEBCHK routine is in *OS/VS2 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.

When a multivolume data set is opened, the direct-access storage devices with the rotational position sensing (RPS) feature incorporate this feature into the channel program only if all of the devices allocated have the record-ready feature.

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 is specified, the executors load the UCS/FCB image specified in the job control statement.

IF UCS/FCB images are not specified in the user's JCL, the executors must insure that the current images, as specified by the UCB UCS extension, are loaded.

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 24 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 24 under Executors.

In Figure 24, 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.

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 blocksize if the buffer length is specified

  that the blocksize 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 blocksize for fixed-length unblocked data sets

  An invalid format of FBS or FS was specified for a partitioned data set.

Access Method Options	Selections															
Actual Data Set	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X
Dummy Data Set							X									
*, DATA, or SYSOUT specified in DD statement													X			
3505 (OMR/RCE) or 3525												X				
3886 (OCR)														X		
3800 (Printer Subsystem)															X	X
Direct Access Device		X		X						X	X					
Printer with UCS Feature (1403, 3203, or 3211)					X											
Printer with forms control buffer (3203 or 3211)						X										
Buffer Pool Required			X	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	1A	1A
IGG0191B	1B	1B	1B	1B	1B	1B		1B	1B	1B	1B	1B		1B	1B	1B
IGG0191C							1C									
IGG0191I			1I	1I							1I	1I		1I		1I
IGG0191N		1N		1N						1N	1N					
IGG0191T					1T	1T										
IGG0191U					1U											
IGG0191V					1V											
IGG0191Y								1Y	1Y	1Y	1Y					
IGG0193I	3I	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	6A	6A
IGG0196B	6B	6B	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	6I	6I
IGG0196Q															6Q	6Q
IGG0196R														6R		
IGG0197E						7E										
IGG0197F						7F										
IGG0197L												7L				
IGG0197M												7M				
IGG0197U					7U											
IGG0199F													9F			
IGG0199G													9G			
IGG0199W													9W			

Figure 24. Open Executor Selector—Stage 1

For BSAM and QSAM:

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

- It performs a test to determine if the blocksize is an integral multiple of the logical record length (LRECL) for QSAM with fixed blocked records or BSAM data sets. If the blocksize 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:

- If the DCB is not a TSO DCB, this routine fills in the DCBDEVT field with the device type and number from the UCB. If unit record equipment is indicated, the routine sets the UR bit in the DCBDEVT field.
- 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:
  a. For direct-access OUTPUT or OUTIN, the track balance of the last Write, from the DSCB, is stored in the DCBTRBAL.
  b. 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 non-zero 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 IGG01911 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 IGG0191V, IGG0197U, IGG0197F, or 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 field DCBBUFNO 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. Eight (4 bytes of padding) more bytes are added to the buffer control block to store the address of the record area. Flags are set (X'C0') to indicate extended buffer control block and presence of record area.
- It issues a DMABCOND macro instruction if BUFTEK=A is specified and the 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 (a) physical sequential in organization, (b) allocated in cylinders, and (c) 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 IGG0191T: Executor IGG0191T is entered after Open executor IGG0196A or IGG0196B if a printer with the UCS feature is specified.

The executor operates as follows:

- It uses the EXCP macro instruction to execute block data check or reset block data check according to the specification in this DCB. If the EXCP fails, an XCTL to IGG0191V is made to get a DMABCOND macro issued.
- It issues a LOAD macro instruction for the message CSECT.
- If the task that issued the OPEN is the communications task, it XCTLs to IGG0191V to find the appropriate stage 2 executor.
- It examines the UCB and JFCB to determine which FCB image or UCS image is to be loaded. If the printer is a 3203 or 3211, and no FCB or UCS activity is required, control is passed to IGG0197F to clean up and pass control to a stage 2 executor.
- When no UCS image is specified in the JFCB and the UCB has no UCS image ID, or the UCS image ID in the UCB is not a default UCS image, the executor requests an operator to specify a UCS image. Then it specifies

in the WTG table that executor IGG0191U is the next executor required for this DCB.

When no UCS image is specified in the JFCB but UCS image ID in the UCB is a default UCS image ID, the currently loaded UCS image is used for this DCB and is 'force loaded' for integrity reasons.

- If no FCB image is specified in the JFCB, but the FCB image ID in the UCB is a default ID, the currently loaded FCB image is 'force loaded' for integrity reasons.
- If no FCB image is named in the JFCB and the UCB has no FCB image ID, or the FCB image ID in the UCB is not a default image, the executor issues a console message requesting that an image be specified.
- The message text for all console messages is extracted from the message CSECT.
- IGG0191U is specified as the next executor.
- It opens SYS1.IMAGELIB and sets an audit trail bit to indicate that SYS1.IMAGELIB is open. If the IMGLIB SVC fails, control is passed to IGG0191V for a DMABCOND macro to be issued.
- It then searches the WTG table to pass control to another executor.

Stage 1 Open Executor IGG0191U: Executor IGG0191U is entered after executor IGG0191T when the specified UCS image is to be loaded from SYS1.IMAGELIB.

The executor operates as follows:

- It requests the operator to mount a chain/train cartridge only if the UCS image specified in the DD statement is different from the currently loaded image.
- The message text for console messages is extracted from the message CSECT.
- It uses the BLDL macro instruction to locate the UCS image in the SYS1.IMAGELIB. If an I/O error occurs during BLDL processing, an XCTL to IGG0191V is made to get a DMABCOND macro issued.
- If the image is not found in the library, the executor requests the operator to specify an alternate UCS image to be used.
- If the operator replies "cancel," it sets an ABEND code for IGG0191V to have a DMABCOND macro instruction issued.
- The executor specifies in the WTG table that IGG0191V is the next executor required for this DCB.

Stage 1 Open Executor IGG0191V: Executor IGG0191V is entered after executor IGG0191U to load the UCS image into virtual storage and subsequently into the UCB buffer. It can also issue a message requesting the operator to specify an FCB image.

The executor operates as follows:

- It tests to see if it was entered to issue a DMABCOND macro and if so, issues the macro.
- It uses the LOAD macro instruction to retrieve the UCS image from SYS1.IMAGELIB.

- It sets an audit trail bit to indicate that a UCS image was loaded and must be deleted.
- It uses the EXCP macro instruction to load the UCS image into the UCS buffer. If an I/O error occurs, a DMABCOND macro instruction is issued.
- When all the following conditions are met, it requests the operator to specify which FCB image is to be loaded:

  The printer is a 3203 or 3211.

  The current FCB image is not a default.

  An FCB image was not specified in the DD statement.

  The UCS image does not have to be verified. (If retrieval of a UCS image from SYS1.IMAGELIB is not required, IGG0191T issues the FCB image request.)
- The message text for console messages is extracted from the message CSECT.

- The executor updates the entry in the WTG table with one of the following:

  IGG0197U if the UCS image must be verified.

  IGG0197E if an FCB load and/or verification is required, and UCS verification is not required.

  IGG0191I if the printer is a 1403 and the buffer control block is specified.

  IGG0191G if the printer is a 1403 and normal scheduling is specified.

  IGG0191Q if the printer is a 1403 and chained scheduling is specified.

  IGG01911 if the DCB specifies EXCP.

- 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 40 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, should be used to determine which stage two executor is required for this DCB.

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 where 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.
- 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 the UCS or FCB feature—executor IGG0191T.

  If the device type is a 3800 printer 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 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 five (three for 2520 or 2540) if not previously specified.
- Executor issues DMABCOND macro instruction (calls problem determination module) if the buffer length is less than blocksize or if data set is for a printer and something other than output (only) is specified.
- Determines the next executor to receive control.

  a. 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 IGG01911.

  b. 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.

  c. If a buffer pool is required, IGG01911 receives control.

  d. If the allocated device is a 3800 printer, the next executor is IGG0196Q.

  e. 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, as 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 the 3800 printer and is entered only when a data set is allocated to a 3800. It receives control from IGG0196A (EXCP) and IGG0196B (BSAM/QSAM).

The executor operates as follows:

- It obtains storage for the 3800 ERP work area if one does not exist. Once obtained, the ERP work area remains until the next system IPL.
- It obtains storage for a SETPRT parameter list. If the JFCB indicates that a JFCBE (JFCB extension for the 3800) 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. The module sets the SETPRT initialization bit on, this causes the SETPRT executors to reset the device to its hardware defaults before the device is set up with data set dependent requirements.
- If SETPRT is not successful, message IEC162 with SETPRT return codes is issued, followed by an ABEND (IEC141I 013-CC).
- 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, should be used.

Stage 1 Open Executor IGG0196R: See *OS/VS IBM 3886 Optical Character Reader Model 1 Logic.*

Stage 1 Open Executor IGG0197E: Executor IGG0197E locates the FCB image and loads the FCB buffer. It is entered from IGG0191T, IGG0191V, or IGG0197U.

The executor operates as follows:

- 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 was not defined in the problem program.
- It issues a LOAD macro instruction to bring the FCB image into storage and sets an audit trail bit to indicate that the FCB image must be deleted.
- If the image is not found in the library, the executor requests the operator to specify an alternate FCB image.
- If the operator replies 'cancel,' it sets a code for IGG0197F to issue a DMABCOND macro instruction.

- The FCB image is loaded into the FCB buffer. If an I/O error occurs during the FCB load, issue a console message.
- It resets the FCB-image-loaded audit trail bit if it deletes the FCB image.
- The message text for console messages is extracted from the message CSECT.
- It specifies in the WTG table that IGG0197F is the next executor required for this DCB.
- It then searches the WTG table to pass control to another executor.

Stage 1 Open Executor IGG0197F: Executor IGG0197F prints a verification of the FCB image and issues an "align forms and verify" message to the operator. It is entered form IGG0191T, IGG0197E, or IGG0197U.

The executor operates as follows:

- It checks to see whether an align-forms-only or a verify-only switch is set.
- If VERIFY is specified, the FCB image is printed.
- If VERIFY or ALIGN is specified, the operator is instructed to align the forms.
- It resets the audit trail bits for FCB image loaded and SYS1.IMAGELIB opened after it deletes the FCB image and closes SYS1.IMAGELIB.

The executor specifies in the WTG table the next module required for this DCB, as follows:

IGG0191I if the buffer control block is specified.
IGG0191G if normal scheduling is specified.
IGG0191Q if chained scheduling is specified.
IGG0191I if the DCB specifies EXCP.

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

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:

  IGG01911 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.

  IGG01911 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 IGG0197U: Executor IGG0197U is entered from IGG0191V to verify a UCS image. It can also issue a message requesting the operator to specify an FCB image.

The executor operates as follows:

- It prints the UCS image.
- It deletes the UCS image, closes SYS1.IMAGELIB, and resets the corresponding audit trail lists.
- It asks the operator to specify the FCB image to load, only if all of the following conditions exist:

  The printer is a 3203 or 3211.

  The current FCB image is not a default.

  An FCB image was not specified in the DD statement.

  The executor specifies in the WTG table the next module required for this DCB, as follows:

  IGG0197E if an FCB load and/or VERIFY is required.

  IGG0191I if the printer is a 1403 and the buffer control block is specified.

  IGG0191G if the printer is a 1403 and normal scheduling is specified.

  IGG0191Q if the printer is a 1403, and chained scheduling is specified.

  IGG01911 if the DCB indicates EXCP.

- 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).

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 *OS/VS2 Data Areas*.
- It constructs an ACB and a 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 38).
- 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 VS 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 25 lists the access conditions that cause the different stage 2 executors to be loaded and to receive control.

The device-oriented processing performed by a stage 2 executor primarily consists of the construction of input/output blocks (IOB), 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 25 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 Figure 25 under Executors.

In this figure an X in a column represents a condition that must be met for the executor to be selected. A No in a column indicates that the condition must not be specified 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 not examined at this time. The table should be used in conjunction with the

Access Method Options[1]	Selections																						
BSAM or	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
QSAM	X	X	X	X		X		X			X	X		X		X	X	X	X	X	X	X	X	
Input or		X	X					X						X		X	X	X	X	X		X	
Output	X													X				X		X	X		
Inout, Outin					X		X						X		X								
Update				No	No	No					X				X							No	
Unit Record or				X								X					X	X	X	X		X	
Magnetic Tape or				X	X							X	X										
Paper Tape				X																			
Direct-Access Storage	X	X	X				X	X			X			X	X						X		
Write-Load (Create-BDAM)									X	X													
Track Overflow	No	No	No	No	No	X				X											X	No	
Chained Scheduling	No	No	No	No	No			X				X	X	X								No	
Search Direct		X																			X		
RPS Device											No				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	
1419 MICR																							X
Executors																							
IGG0191D	1D	1D	1D																				
IGG0191G				1G	1G											1G	1G				1G		
IGG0191H						1H																	
OGG0191J							1J																
IGG0191K								1K															
IGG0191L									1L	1L													
IGG0191M										1M													
IGG0191O			1O																				
IGG0191P											1P				1P								
IGG0191Q												1Q											
IGG0191R													1R										
IGG0191S						1S														1S			
IGG0191W														1W									
IGG0191X															1X								
IGG0191Z																1Z							
IGG019123																23							
IGG0196K																					6K		
IGG0196L							6L																
IGG0196P											6P												
IGG0196S[2]																						6S	
IGG0197C[3]																							7C
IGG0197D[3]																							7D
IGG0197N																	7N	7N	7N	7N			
IGG0197P																			7P	7P			
IGG0197Q																			7Q	7Q			
IGG0197V																						7V	
IGG0199K																					9K		
IGG0199L									9L														
IGG0199O		9O																					

1. If *, DATA, or SYSOUT are specified on the DD statement, no stage 2 executors are loaded.
2. See OS/VS TCAM Logic, SY30-2059.
3. See OS BSAM Logic for IBM 1419/1275, GY21-0012.

Figure 25. Open Executor Selector—Stage 2

flow of control information in Diagram E, SAM Flow of Control for Open Executors.

Stage 2 Open Executor IGG0191D: Executor IGG0191D receives control after executors IGG0196B, IGG0193I under normal conditions or from executors IGG0191W, IGG0191K (chained scheduling not supported), under abnormal conditions if the Open parameter list specifies:

Input or Output

and the DCB specifies:

Direct-access storage device

BSAM or QSAM and simple buffering

However, track overflow, and chained channel-program scheduling are not specified.

The executor operates as follows:

- It calculates the amount of virtual storage required and issues a GETMAIN macro instruction to get the space from subpool 0 in the user's key for IOBs and associated channel programs. 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.
- If input is specified with search direct (OPTCD=Z), control is passed to IGG0199O where the channel program is constructed.
- If input is specified without search-direct, control is passed to IGG0191O where the channel program is constructed.
- For output data sets, the executor constructs IOBs and write-channel programs. The address of the first IOB is placed in the DCB. See Appendix B for the format of the channel program constructed by this executor.
- It issues a DMABCOND macro instruction if buffers are not available for DCBs that specify QSAM.

A test of the non-rotational position sensing (RPS) indicator bit is made to see whether the channel programs utilize the RPS feature. If the bit is on (1), standard channel programs are built. If, however, the bit is not on, additional virtual storage is acquired to employ the RPS feature's two CCWs (Set-sector and Read-sector) in the channel programs. The two commands are incorporated where appropriate.

- If variable-length records are specified, IGG01915 is the next executor required for this DCB. Otherwise IGG01910 is specified in the WTG table as the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191G: Executor IGG0191G receives control after executors IGG0196B, IGG01931, IGG0191V, IGG0196Q, IGG0197F, IGG0197U, IGG0197N under normal conditions or from executors IGG0191R, IGG0191Q (chained scheduling not supported), under abnormal conditions if:

The DCB specifies BSAM or QSAM and either unit record, magnetic tape, or paper 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 paper tape, the next executor is IGG01912. 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 IGG0191H: Stage 2 Open executor IGG0191H receives control after executor IGG0191S, if the DCB specifies:

Track overflow

(but not update). If both track overflow and update are specified, executor IGG0191P receives control.

The executor operates as follows:

- It constructs IOBs and associated channel programs, using the storage gotten by IGG0191S.
- It issues a DMABCOND macro instruction if buffers are not available for DCBs that specify QSAM.
- The module checks the non-rotational position sensing (RPS) indicator and, if it is off, inserts the RPS CCWs. When RPS channel programs are built for variable record format, the SLI bit is turned on in the Read-data CCW, thereby eliminating length checking.
- It identifies the end-of-block routine and the direct access Note Point routine to be used in the processing specified by this DCB.

- It specifies in the WTG table that executor IGG01913 (for IGG01916, if the DCB specifies variable-length record format) is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191J: Executor IGG0191J receives control after executor IGG0196B or IGG0193I under normal conditions or IGG0191X if chained scheduling was requested but could not be honored and if the Open parameter list specifies:

INOUT or OUTIN

and the DCB specifies:

Direct-access storage

The executor operates as follows:

- It calculates the amount of virtual storage needed to build the IOBs for the data set and then issues a GETMAIN from subpool 0, in the user's key, for the required space. It then sets the area to zeros. It stores the number of bytes gotten for the IOBs in the second word of the audit trail for force close.
- In calculating the virtual storage area needed for the IOBs, the executor tests for non-rotational position sensing. If the indicator is off, additional space to implement the RPS CCWs in the channel programs is acquired.
- When control is returned from GETMAIN, it sets an audit trail bit to indicate to the Force Close executor that storage should be freed.
- The executor then begins constructing the channel programs and filling in fields in the IOBs. It constructs the search ID equal, the TIC, the READ, and if RPS is specified, the Set-sector commands. It also includes a portion for write-check if specified.
- The executor specifies in the WTG table that executor IGG0196L is the next executor needed for this DCB and then searches the WTG table to pass control to another executor

Stage 2 Open Executor IGG0191K: Executor IGG0191K receives control after executor IGG0196B or IGG0193I if the DCB specifies:

Chained channel-program scheduling

Direct-access storage

Open parameter list specifies INPUT

It is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- If the NOTE/POINT macro instruction is used, the executor identifies direct access Note/Point module IGG019BK to be loaded for use with this DCB.
- It identifies the end-of-block routine to be loaded and used for the processing described by this DCB.
- It sets the ID number in DCBCNTRL field.
- From subpool 0, in the user's key, it obtains space for and constructs one IOB, the required number of ICBs (that is, one ICB per channel program

or buffer) and their associated channel programs, and then links them. It 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.
- A test of the non-rotational-position-sensing (RPS) indicator (bit 2 of JFCBMASK+6) is made. If this bit is not on, additional virtual storage is acquired by GETMAIN to incorporate the RPS CCWs. An additional doubleword is also acquired for the sector values. When the channel programs are built, the new CCWs are inserted.
- It issues a DMABCOND macro instruction if buffers are not available for DCBs that specify QSAM.
- It sets the PCI flag in the Read Count CCW, only if in a real address environment.
- It sets bit 5 of DCBCIND2 to one to indicate that chained scheduling is being supported.
- If the record format is variable length spanned, the executor specifies IGG01916 as the next executor to receive control; otherwise, IGG01913 is specified as the next executor to receive control.
- If the number of channel programs is less than or equal to one (DCBBUFNO if QSAM or DCBNCP if BSAM); the executor sets bit 5 of DCBCIND2 to zero to indicate that chained scheduling is being not supported. It then specifies IGG0191D as the next executor to receive control.
- It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191L: Executor IGG0191L receives control after executor IGG0196B or 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 issue 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 IGG0191O: Executor IGG0191O receives control from IGG0191D if the Open parameter list specifies:

Input

and the DCB does not specify:

Search Direct (OPTCD=Z)

The executor constructs Read channel programs for the IOBs constructed in IGG0191D.

The module tests the non-rotational position sensing (RPS) indicator. If the indicator is not on, IGG0191O inserts the RPS CCWs, where appropriate, in the channel program.

For QSAM DCBs it issues a DMABCOND macro instruction if buffers are not available.

The Read channel program is modified for offset Read (that is, for reading a BDAM data set with VS record format and keys using BSAM READ macro instructions.)

- If the record format is variable, IGG01915 is specified as the next executor to receive control; otherwise, IGG01910 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 IGG0191P: Stage 2 Open executor IGG0191P receives control after executors IGG0196B or IGG0193I if the Open parameter list specifies:

Update

(whether or not track overflow is also specified). It is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- It identifies module IGG019CC as the end-of-block routine to be loaded for use with the DCB.
- If the NOTE/POINT macro instruction is specified, it identifies module IGG019BK as the NOTE/POINT routine to be loaded for use with this DCB.
- It calculates the amount of storage required and issues a GETMAIN macro instruction to get the space from subpool 0 in the user's key for IOBs and associated channel programs. 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.
- Calculations for the amount of storage needed include one byte for all Read channel program segments and one byte for each Write channel program segment. The total number of extra bytes is equal to the number of IOBs, plus one. This value is rounded up to a multiple of eight and added to the total.
- If RPS, executor IGG0191Z is specified in the WTG table. If non-RPS, executor IGG0196P is specified in the WTG table. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191Q: Executor IGG0191Q gains control after executors IGG0196B, IGG0191V, IGG0196Q, IGG0197U, IGG0197F, or IGG01931 if the DCB specifies:

Chained channel-program scheduling

Unit record, 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, 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, unless variable spanned record format is specified. In this case, IGG01916 is specified as 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 executors IGG0196B or 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 direct-access storage, it identifies Note 'Point module IGG019BK to be loaded for use with the DCB.
- If the device is magnetic tape, it identifies Note/Point module IGG019BL to be loaded for use with the DCB.
- It identifies the end-of-block routine to be loaded for use with the DCB.
- 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 IGG0191S: Stage 2 Open executor IGG0191S receives control after executor IGG0196B or IGG0193I if the DCB specifies:

Track overflow

(but not update). The executor is loaded and gains control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It identifies the end-of-block routine and the direct-access NOTE/POINT routine to be used in processing specified by this DCB.
- From subpool 0, in the user's key, it obtains space for and constructs IOBs and channel programs for the maximum number of segments possible. For RPS devices, it increases the space required by the amount necessary to implement two RPS CCWs. It links the channel programs to the IOBs and the IOBs to one another. 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.
- If search-direct has been requested (OPTCD=Z in the DCB), the executor specifies IGG0199K as the next executor required for this DCB.
- It specifies in the WTG table that executor IGG0191H is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191W: Executor IGG0191W receives control after executor IGG0191B or IGG0193I if the DCB specifies:

Chained channel-program scheduling

Direct-access storage

Output

The executor operates as follows:

- 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 (that is, one ICB per channel program or buffer) and their associated channel programs, and then links them. It stores the number of bytes gotten for the IOBs in the second word of the audit trail for force close. If the non-rotational position sensing (RPS) bit is off, the space for the IOB and ICB is increased to incorporate the RPS CCWs and the space is inserted.
- 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.
- It issues a DMABCOND macro instruction if buffers are not available for DCBs that specify QSAM.

- 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 IGG0191D as the next executor to receive control. If bit 5 of DCBCIND2 is set to 1 to indicate support of chained scheduling and IGG01916 is specified as the next executor to receive control unless the record format is variable length spanned, the next executor to receive control is IGG01913.
- It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191X: Executor IGG0191X receives control after executors IGG0191B or IGG0193I if the Open parameter list specifies:

INOUT or OUTIN

and the DCB specifies:

Chained scheduling

Direct-access storage

The executor is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- 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 and links them. It stores the number of bytes gotten for the IOBs in the second word of the audit trail for force close. If the rotational position sensing (RPS) indicator is off, the space acquired for the IOB is incremented to incorporate the RPS CCWs, which will then be inserted in the channel program.
- 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 IGG0191J as the next executor to receive control; otherwise, bit 5 of DCBCIND2 is set to 1 and IGG01913 is specified as the next executor to receive control, unless variable spanned record format is specified. In this case, IGG01916 is the next executor for this DCB.
- It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191Z: Executor IGG0191Z receives control after executor IGG0191P, if the Open parameter list specifies:

Update

and:

Record-ready channel programs are to be generated

The executor operates as follows:

- It begins to construct IOBs and channel programs to empty and refill each buffer.
- It links the IOBs and sets read only bits.
- It sets ECBs to X'7F' and initializes IOB DCB pointers.
- It builds the empty portion of the channel program; the write check and track overflow portion (if user specified), and begins building the refill portion of the channel program.
- It stores the offset to the write channel program, the write channel program length, and (for QSAM) the offset to the read CCW into the DCB.
- It saves, in the IOB, the buffer addresses associated with the empty portion of the channel program. These addresses are required by the next executor when building the refill portion of the channel program.
- It then saves displacements to indicate to IGG01923 where it is to begin constructing the remainder of the channel programs and then passes control to IGG01923.

Stage 2 Open Executor IGG01923: Executor IGG01923 receives control after executor IGG0191Z has completed constructing its portion of the IOB and channel programs.

The executor operates as follows:

- It finishes building the channel programs started by IGG0191Z and completes initialization of the IOBs.
- It then passes control to IGG01915 if the record area indicator is on in the buffer control block. If the record area indicator is off, it passes control to IGG01912.

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 IGG0196L: The executor receives control from executor IGG0191J to finish building the IOBs assembled in IGG0191J.

The executor operates as follows:

- Starting at the end of the last CCW constructed by IGG0191J, it completes the building of the channel programs. Appendix B, "BSAM/QSAM Channel Programs," shows the channel program constructed by this executor and executor IGG0196L.

- For search direct it alters the channel program to be Search, TIC, Read Count, Read Data, Read Count instead of Search, TIC, TIC, Read Data, Read Count, to be positioned correctly to read the first record. This channel program is the one associated with the first physical IOB. The channel program is restored to normal by the channel-end appendage at channel-end time.
- The executor specifies in the WTG table that executor IGG01910 (or IGG01915, if the DCB specifies variable-length record format) is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0196P: IGG0196P receives control after executor IGG0191P, if the OPEN parameter list specifies:

Update

and:

Non-record-ready channel programs are to be generated

The executor operates as follows:

- It constructs IOBs and channel programs to empty and refill each buffer.
- For QSAM, the executor links the channel programs so that a buffer may be either refilled only (by executing only the second half of the channel program) or emptied and refilled (by executing the channel program from the beginning).
- If record area is present (which indicates that the record format is variable-length spanned), it specifies in the WTG table that executor IGG01915 is the next executor required for this DCB. Otherwise, it specifies executor IGG01912. It 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.
- Once 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 IGG0199K: Executor IGG0199K receives control after executor IGG0191S.

The executor operates as follows:

- Using the virtual storage gotten by IGG0191S, it constructs the IOB and channel programs for direct-access devices with the search-direct feature (OPTCD=Z).
- It issues a DMABCOND macro instruction for QSAM DCBs if buffers are not available.

- It changes the first channel program to be executed (the second physical IOB) to Search, TIC, Read Count, Read Data, Read Count, from Search, TIC, TIC, Read Data, Read Count, so that it is positioned correctly to read the first record. The channel program is changed back to normal by the channel end appendage.
- Completes related fields in DCB.
- It specifies in the WTG table that IGG01916 is the next executor required for the DCB if variable-length records are specified. Otherwise, IGG01913 is specified in the WTG table.
- It then searches the WTG table to pass control to another executor.

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 2 Open Executor IGG0199O: IGG0199O receives control from executor IGG0191D if the Open parameter list specifies:

Input

and the DCB specifies:

OPTCD = Z (search-direct)

The executor operates as follows:

- Using the core gotten by IGG0191D, the executor constructs the IOBs and channel programs required when search direct is specified. The format of the channel programs constructed by this executor are shown in "Appendix B: BSAM/QSAM Channel Programs."
- It changes the channel program to Search, TIC, Read Count, Read Data, Read Count, from Search, TIC, TIC, Read Data, Read Count, so that it is positioned correctly to read the first record. (This channel program is the one associated with the second physical IOB.) The channel program is restored to normal by the channel end appendage.
- It issues a DMABCOND macro instruction for QSAM DCBs if buffers are not available.

- If format-F or -U records are specified, IGG01910 required for this DCB. Otherwise, (for format-V) executor IGG01915 is specified in the WTG table as the next executor required for this DCB.
- It then searches the WTG table to pass control to another executor.

Stage 3 Open Executors

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. The four 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 four bytes and the modules they can identify, depending on which stage 3 executor does the loading, are:

00	No module to load	
01	IGG019CD, IGG019CV	End-of-block
02	IGG019CC, IGG019CW	End-of-block
03	IGG019CE, IGG019CX	End-of-block
04	IGG019CF, IGG019CY	End-of-block
05	IGG019BC, IGG019BK	NOTE/POINT, DASD
06	IGG019BD, IGG019BL	NOTE/POINT, tape
07	IGG019CA, IGG019BC	CNTRL, card reader or NOTE/POINT, DASD
08	IGG019CB, IGG019CC	CNTRL, printer or End-of-block
09	IGG019BE, IGG019C2	CNTRL, tape or EOB, track overflow
0A	IGG019AV	DUMMY or no-op for various functions
0B	IGG019CT, IGG019TD	End-of-block, error or user-totaling
0C	IGG019TD, IGG019TC	End-of-block, user-totaling
0D	IGG019TC, IGG019TV	End-of-block, user-totaling
0E	IGG019TV, IGG019TW	End-of-block, user-totaling
0F	IGG019TW, IGG019T2	End-of-block, user-totaling
10	IGG019T2, IGG019CT	End-of-block, user-totaling or error

In many of the above pairs the first one is for normal scheduing 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 26 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 26.

In this table an X 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 IGG0191D, IGG0191O, IGG0197Q, IGG0199O or IGG0196L. It also receives control after executor IGG0191G unless the DCB specifies paper tape.

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
- 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 IGG01917 in the WTG table as the next executor to receive control for this DCB.

Stage 3 Open Executor IGG01911: Executor IGG01911 is entered from executors IGG0191C, for dummy data sets; IGG0191N, IGG0191V, IGG0196A, IGG0197U, and IGG0197F for EXCP data sets; and IGG01917, IGG01918, IGG01926, IGG01993, and IGG01994 for all SAM, PAM, and BDAM CREATE data sets.

Get or Put

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.

Access Method Options	Selections									
Paper Tape	X									
Update		X								
Chained Scheduling			X							
Track Overflow				X			X			
None of the preceding					X					
*, DATA, or SYSOUT specified on DD statement									X	
Variable-length Record Format						X	X			
Spanned Records								X		
Dummy Data Set										X

Executors										
IGG01910					10					
IGG01911	11	11	11	11	11	11	11	11		11
IGG01912	12	12								
IGG01913			13	13						
IGG01915						15		15		
IGG01916							16			
IGG01917					17					
IGG01918	18	18								
IGG01919			19	19				19		
IGG01926			26	26				26		
IGG0198L									8L	
IGG01991						91				
IGG01992							92			
IGG01993						93				
IGG01994								94		

Figure 26. Open Executor Selector—Stage 3

- It sets the request type as follows:

Processing Required	Type I	Type II	Type III
ERP Processing		X	
CE Appendage		X	
CE Interrupt			X
SIO Appendage	X		
EOE Appendage	X		

- 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 IGG01912: Executor IGG01912 is entered after executor IGG01923, IGG0196P, and IGG0191G if the Open parameter is:

Update

or if the DCB specifies:

Paper tape

The executor operates as follows:

- It identifies and loads all the appendages required and places their addresses in the appendage vector table.
- If rotational position sensing (RPS) channel programs are constructed and the record format is fixed, the format-F channel-end appendage will always be loaded. For RPS with format-U without track overflow, a format-U channel-end appendage is loaded.
- It issues a CIRB macro to build an IRB for UPDATE with track overflow.
- It loads the device-dependent routines.
- It enters the address of a paper tape conversion routine into the DCB, and the address of the paper tape appendage into the appendage vector table.
- It issues a DMABCOND macro instruction if paper tape and unlike attributes are specified.
- It specifies executor IGG01918 in the WTG table as the executor to receive control next for this DCB.

Stage 3 Open Executor IGG01913: Executor IGG01913 receives control after executors IGG0191H, IGG0191K, IGG0191Q, IGG0191X, IGG0191W, IGG0199K, and IGG0191R if the DCB specifies:

Chained channel-program scheduling, or track overflow.

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 specifies in the WTG table that Open executor IGG01919 is to receive control next for this DCB.

Stage 3 Open Executor IGG01915: Executor IGG01915 receives control after executors IGG0191D, IGG0191O, IGG0191G, IGG0196K, IGG0196L, IGG0197Q, and IGG0199O, if the DCB specifies:

Variable-length record format

Executor IGG01915 receives control from executor IGG0196P or IGG019123, if the DCB specifies:

Variable-length spanned 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 issues a DMABCOND macro instruction if LRECL=X is specified and the processing mode is not Locate.
- 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 IGG01991 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 IGG0191H, IGG0191K, IGG0191O, IGG0191Q, IGG0191W, IGG0199K, and IGG0191R if the DCB specifies:

Variable-length record format

Track overflow

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 issues a DMABCOND macro instruction if LRECL=X is specified and the processing mode is not Locate.
- 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 IGG01992 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 IGG01917: Executor IGG01917 is entered after executor IGG01910.

The executor operates as follows:

- It identifies and loads all the appendages required and places their addresses into the appendage vector table.
- If rotational position sensing (RPS) channel programs are constructed and the record format is fixed, the format-F channel-end appendage is always loaded. It loads in the fixed standard end-of-extent module IGG019C4 where the fixed standard record format is used.
- For RPS with format-U without track-overflow, a format-U channel-end appendage is loaded.
- It loads the end-of-block routine identified by a stage 2 executor and places its address into the DCB.
- If search-direct has been requested (OPTCD=Z in the DCB), the executor loads in the necessary appendages.
- It enters into the DEBSUBID field of the DEB the identification of each routine loaded.
- It specifies in the WTG table that executor IGG01911 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 3 Open Executor IGG01918: Executor IGG01918 is entered after executor IGG01912. It is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- It loads the end-of-block routine identified by a stage 2 executor and places its address into the DCB.
- It enters into the DEBSUBID field of the DEB the identification of each routine loaded.

- It specifies in the WTG table that executor IGG01911 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 3 Open Executor IGG01919: Executor IGG01919 is entered after IGG01913.

The executor operates as follows:

- It identifies and loads all the appendages required and places their addresses into the appendage vector table.
- If rotational position sensing (RPS) channel programs are constructed and the record format is fixed, the format-F channel-end appendage is always loaded. For RPS with format-U without track overflow, a format-U channel-end appendage is loaded.
- For track overflow it issues a CIRB macro instruction to create an IRB for an asychronous error processing routine.
- If search-direct has been requested (OPTCD=Z in the DCB), the executor loads in the necessary appendages.
- It enters into the DEBSUBID field of the DEB the identification of each routine loaded.
- It specifies in the WTG table that executor IGG01926 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 3 Open Executor IGG01926: Executor IGG01926 is loaded and receives control after executor IGG01919.

The executor operates as follows:

- It receives control after it is loaded.
- It loads the end-of-block routine identified by a stage 2 executor and places its address into the DCB.
- It enters into the DEBSUBID field of the DEB the identification of each routine loaded.
- It specifies in the WTG table that executor IGG01911 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

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 handles the CHECK macro instruction also.
- 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.

Stage 3 Open Executor IGG01991: Executor IGG01991 receives control after, and as a continuation of, executor IGG01915. It completes the loading of subroutines for a DCB which specifies:

Variable-length or record format-D

The executor operates as follows:

- It identifies and loads all the appendages required and places their addresses into the appendage vector table.
- If rotational position sensing (RPS) channel programs are constructed and the record format is variable, the format-V channel-end appendage is always loaded. It loads in the variable standard end-of-extent module, IGG019C4, where the variable record format is used.
- For RPS with format-U without track overflow, a format-U channel-end appendage is loaded.
- For track overflow, it issues a CIRB macro instruction to create an IRB for an asychronous error routine.
- If search direct has been requested (OPTCD=Z in the DCB), the executor loads in the necessary appendages.
- It enters the IDs of the routines loaded into the DEB subroutine ID field.
- The executor specifies in the WTG table that IGG01993 is the next executor required for this DCB. It then searches the WTG table to determine the next executor to receive control.

Stage 3 Open Executor IGG01992: Executor IGG01992 receives control after, and is a continuation of, executor IGG01916. The executor loads subroutines for a DCB which specifies:

Variable-length record format

Track overflow

The executor operates as follows:

- It identifies and loads all the appendages required and places their addresses into the appendage vector table.
- If rotational position sensing (RPS) channel programs are constructed and the record format is variable, the format-V channel end appendage is always loaded. It loads in the variable standard end-of-extent module, IGG019C4, where the variable-record format is used.

- For RPS with format-U without track overflow, a format U channel end appendage is loaded.
- If search-direct has been requested (OPTCD=Z in the DCB), the executor loads in the necessary appendages.
- For track overflow, it issues a CIRB macro instruction to create an IRB for an asynchronous error routine.
- It enters the IDs of the routines loaded into the DEB subroutine ID field.
- It specifies in the WTG table that executor IGG01994 is the next executor required for this DCB. It then searches the WTG table to determine the next executor to receive control. If there are no other entries in the WTG table, the executor returns control to the Open routine.

Stage 3 Open Executor IGG01993: Executor IGG01993 is a continuation of executor IGG01991. It completes the process of loading subroutines for a DCB that specifies:

Variable-length record format

The executor operates as follows:

- It identifies and loads all of the appendages required and places their addresses into the appendage vector table.
- If rotational position sensing (RPS) channel programs are constructed and the record format is variable, the format-V channel-end appendage is always loaded. It loads in the variable standard end-of-extent module, IGG019C4, where the variable-record format is used.
- It loads the end-of-block routine identified by the stage 2 executor and places its address into the DCB.
- If search-direct has been requested (OPTCD=Z in the DCB), the executor loads in the necessary appendages.
- It enters the IDs of the routines loaded into the DEB subroutine ID field.
- It specifies in the WTG table that executor IGG01911 is the next executor required for this DCB. It then searches the WTG table to determine the next executor to receive control.

Stage 3 Open Executor IGG01994: Executor IGG01994 is loaded and receives control from IGG01992. It completes the process of loading subroutines for a DCB that specifies:

Variable-length record format

Track overflow

The executor operates as follows:

- It loads the end-of-block routine identified by the stage 2 executor and places its address into the DCB.
- It enters the IDs of the routines loaded into the DEB subroutine ID field.
- It specifies in the WTG table that executor IGG01911 is the next executor required for this DCB. It then searches the WTG table to determine the next executor to receive control.

Close Executors

Figure 27 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 CPU 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		
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 27. 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 2520 or 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 seven.
- 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, it issues EXCP and WAIT macro instructions to clear the 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 (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*
Read/Punch/Print	Read File**
Read/Punch	Read File**
Punch/Print	Punch File
Punch/Interpret	Punch File
Read	Read File
Punch	Punch File

* 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.

** Punching or printing delimiter cards is not allowed for these file types since 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 frees the FCR table for the IBM 3886 Optical Character Reader.
- 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 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 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 (MEMBERNAME) are specified, the executor issues a STOW macro instruction. On completion of the Stow routine, the executor tests for I/O errors and for logical errors, such as insufficient space in the directory. For either type of error, the executor issues 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 pre-open 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 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 users 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 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 Area (CRSA) with a return code of eight 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 eight in register 15.

Buffer-Pool Management

Buffer-pool management routines form virtual storage space into buffers, and they return buffers that are no longer needed. Figure 28 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 28. 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 eight 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 29 illustrates the buffer-pool control block (BUFCB) that describes the buffer pool. Figure 30 illustrates the buffer-pool structures formed by the GETPOOL module.

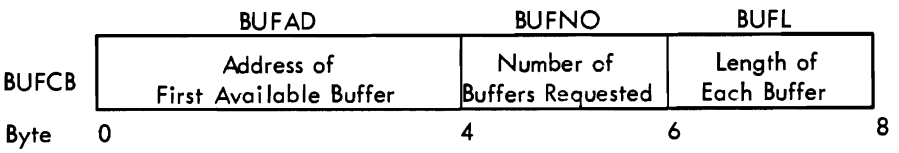


Figure 29. Buffer-Pool Control Block

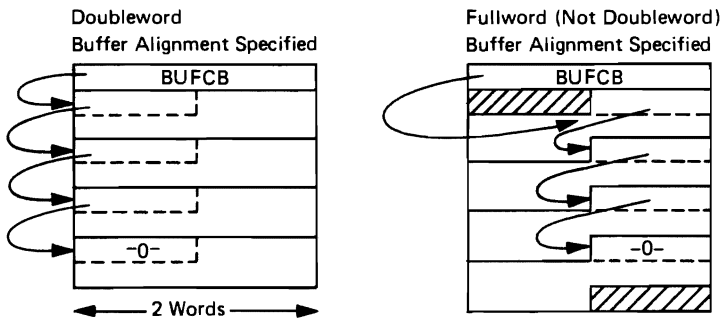


Figure 30. GETPOOL Buffer-Pool Structures

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 31 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 29 illustrates the buffer pool control block (BUFCB), Figure 32 illustrates the various buffer alignments that the Build module forms.

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

Figure 31. Build Buffer-Structuring Table

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.

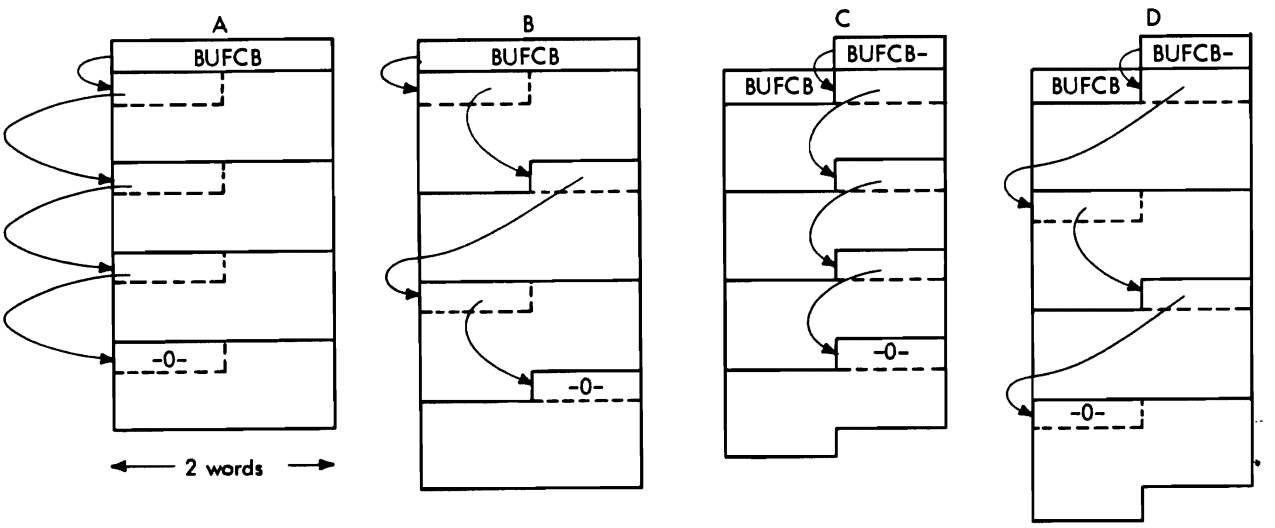


Figure 32. Build Buffer-Pool Structure

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.

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) in the first twelve 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) 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 33 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 34 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.

BUFAD	BUFLG	BUFNO	BUFLTH	BUFRECAD
Address of First Available Buffer	Flags	Number of Buffers Requested	Length of Each Buffer	Address of Record Area
Byte 0	4	5	6	8	12

Figure 33. Logical Record 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.

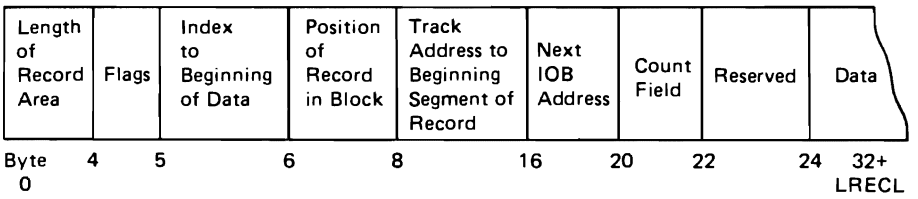


Figure 34. Record Area

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 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. 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 which has a beginning segment of a record that is being processed. The low-order three 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.

- **Reserved.** Not used.
- **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 core 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 a 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 core.

- 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 and the Device Characteristics Table for the required information.
- It places the data in the output area and returns to the calling program.
- For the IBM 3340 Direct Access Disk Storage Facility, it determines the number of cylinders on the pack.

IMGLIB Routine

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 0s, 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 Routines

Control Module IGC0002E (SVC 25—Track Balance, Track Overflow Erase): Module IGC0002E consists of two routines that erase either a part of one track or several tracks. The track balance routine determines the available space by erasing the remainder of the track; the track-overflow erase routine erases tracks at the end of each extent on which there are no data fields for blocks of the data set to which the extent belongs. The routine is used when a block in a data set with track-overflow record format would span extents.

This module is entered if Read/Write module IGG019BA, End-of-Block module IGG019C2, or Close executor IGG0201B issues an SVC 25 instruction.

- It issues an ESTAE macro instruction to establish a TRR, IGCT002E, to intercept abnormal terminations.

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

The routine operates as follows:

- It constructs and issues an EXCP macro instruction for a channel program with the Erase command and a count exceeding the track capacity. The erase operation begins following the block just read or on the block pointed to.
- It determines the approximate track balance by subtracting the residual count in the channel status word (CSW) from the count used in the channel program and inserts the difference in the DCBTRBAL field of the DCB.
- If standard format is specified and the DCB blocksize is equal to the blocksize saved at Open time, the track balance will be computed arithmetically (rather than taking the approximation from the ERASE operation) and stored in the DCBTRBAL field.

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

The routine operates as follows:

- It receives control when it is loaded.
- It substitutes Erase commands for the Write commands in the channel program associated with the present IOB.
- It issues an EXCP macro instruction to cause execution of the channel program and a WAIT macro instruction for its completion.
- It returns control to the track-overflow end-of-block routine, irrespective of any errors in the execution of the channel program.

Control Module IGC0006I (SVC 69—BSP): Module IGC0006I backspaces the data set one block whether the data set is on a magnetic-tape or direct-access device.

The expansion of the macro instruction BSP includes an SVC 69 instruction which causes the module to be loaded and entered.

The module 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 preceding DCBFDAD value has been established, the module returns control to the processing program with register 15 set to zero.
- If there is no valid preceding DCBFDAD value because the processing program 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 in 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.

BLDL or FIND Routines

FIND (C Option) Macro Expansion: The macro expansion moves the relative address (TTRK) from the BLDL list in virtual storage to the DCBRELAD field in the requester's DCB. The FIND macro instruction then does a branch-and-link to the Point routine.

Resident Module IGC018: At initial program loading (IPL) time, the nucleus initialization program (NIP) constructs a resident BLDL table from SYS1.LINKLIB directory entries. That table is the one referred to by the Find and BLDL routines in this module.

The routines comprising the module gain 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, the entry point for the Find (D Option) and BLDL routines. Control programs may use a BALR instruction and the address found in the communications vector table (CVT) for entry points IECPCNVT, IECPRLTV, and IEC0SCR1 to pass control to the respective routines.

Find (D Option) Routine—Entry Point and CSECT Name: IGC018 (SVC 18): The Find (D Option) routine finds the relative address of the member named in the macro instruction. It then causes the relative address to be converted into the full direct-access device address (FDAD) and to be loaded into the DCBFDAD and IOBSEEK fields.

The routine operates as follows:

- It issues an ESTAE macro instruction to establish the TRR, IGCT0018.
- If SYS1.LINKLIB is the referenced library, it scans the resident BLDL table for an entry that matches the given member name.
- If SYS1.LINKLIB is not the referenced library, or if the name is not in the table, it searches the directory for an entry block with a key equal to or higher than the given member name. It reads that entry block into virtual storage and searches the entry block for the matching entry.
- If the name is in the table, or after finding the matching entry in an entry block read in, it enters the relative address stated in the entry into the DCBRELAD field in the DCB.
- It passes control to the Point routine by issuing a BAL instruction for supervisor key callers or a SYNCH macro instruction for user key callers.
- It returns control to the processing program.

BLDL Routine—Entry Point: IGC018 (SVC 18): The BLDL routine completes a BLDL table with the directory entry for each of the members named in the BLDL table.

The routine operates as follows:

- It issues an ESTAE macro instruction to establish the TRR, IGCT0018.
- If SYS1.LINKLIB is the referenced library, it scans the resident BLDL table for an entry that matches the given member name.
- If SYS1.LINKLIB is not the referenced library, or if the name is not in the table, it searches the directory for an entry block with a key equal to or higher than the given member name. It reads that block into virtual storage and searches the entry block for the matching entry.
- If the name is in the table, or after finding the matching entry in an entry block read in, it moves the entry into the processing program's BLDL table, obtains the next name to be matched, and returns to the beginning of the routine.
- When the BLDL table has been completed, the routine returns control to the processing program.

Convert Relative-to-Full Address Routine—Entry Point: IECPCNVT: Conversion routine IECPCNVT accepts, in register 0, relative addresses of the form TTR for direct-access devices and presents the corresponding full device addresses of the form MBBCCHHR at the location shown by

register 2. This routine's external interface is documented in *OS/VS2 System Programming Library: Data Management*.

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 TTR in register 0.

The routine totals the number of tracks per extent for the (M-1) extents. For extent M, it adds the number of tracks entered into the extent. This routine's external interface is documented in *OS/VS2 System Programming Library: Data Management*.

Convert Record Number to Sector Value Routine—Entry Point: IEC0SCR1: Conversion routine IEC0SCR1 converts the record number for a fixed or variable length record data set to a sector value for use on an RPS device.

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 non-keyed 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 three 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
- Keylength
- 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 which 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. See *OS/VS2 System Programming Library: Data Management* for information about the TRKCALC macro.

SYNADAF and SYNADRLS Routines

See Diagram O 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 TRR (task recovery routine) 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 CI (compatability interface) 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 3211 Printer 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 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 and so the routine increments 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 IOB sense bytes on a unit check condition for direct access, magnetic tape, and unit record devices, except for the 3203, 3211 and 3800 printers. 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 2314 or 3330 device, it stores the write-inhibit message text.
- If the unit record device type is a TCR (tape cartridge reader), the routine re-analyzes 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 and 3800 printer devices.

IGC0806H receives control through a branch from IGC0106H for 3211 Printers 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 and SETDEV Routines

There are 13 routines associated with the SETPRT macro instruction. Routine IGC0008A receives control when the SVC 81 instruction is issued. Routine IGG08101 receives control after routine IGC0008A if a specified UCS image is to be loaded from the SYS1.IMAGELIB. Routine IGG08102 receives control after routine IGG08101 to load the UCS image into the UCS buffer and to print verification lines if required. Routines IGG08103 and IGG08104, respectively, locate the FCB image and load it into the forms control buffer. Routine IGG08104 also verifies the load and/or allows forms alignment.

When the device for which a SETPRT has been issued is a 3800 printer, executor IGC0008A gives control to executor IGG08110. Executors IGG08111, IGG08112, IGG08113, IGG08114, and IGG08115 are given control, depending on the contents of the SETPRT parameter list.

When a SETPRT is issued for a SYSOUT data set, executor IGC0008A gives control to executor IGG08117.

All messages issued by the SETPRT routines are in a message CSECT. The SETPRT routines 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 routines are associated with the SETDEV macro instruction. The SETPRT routine, IGC0008A, receives control when the SVC 81 instruction is issued. Routine IGG08108 receives control from routine IGC0008A to initialize the IBM 3890 Document Processor Control Unit or the IBM 3886 Optical Character Reader.

Figure 34.1 shows the conditions that cause the executors to gain control. 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. Routine IGC0008A determines whether the specified UCS image is to be loaded from the SYS1.IMAGELIB after processing all outstanding output requests for the DCB. It also determines if the 3890 Document Processor Control Unit is to be loaded and, if so, gives control to routine IGG08108. If the 3800 printer is to be set up, control is given to IGG08110. If a SYSOUT data set is requested, control is given to routine IGG08117.

Input to the SETPRT executors is a 56-byte parameter list pointed to by register 1. See the "Data Areas" section for a description of the parameter list.

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.

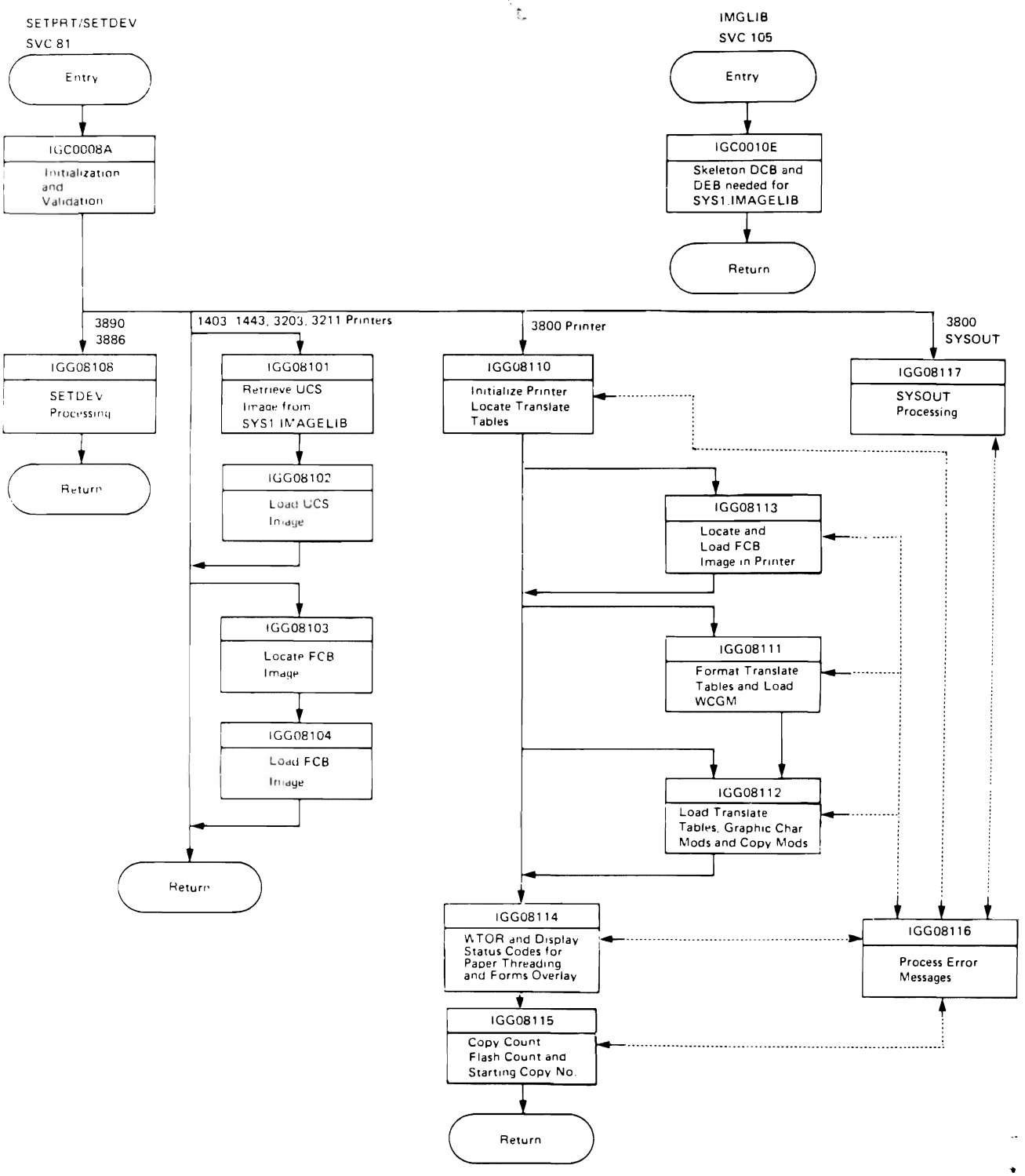


Figure 34.1. SETPRT Executor Selector

- 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 that is open for output and bypasses unit record processing.
- If the 3890 or 3886 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, 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 macro instruction.
- If the user does not specify an image in the SETPRT parameter list, UCS/FCB images will, for integrity reasons, be 'force loaded' with the current images as specified in the UCB.
- It issues SVC 105 to open SYS1.IMAGELIB.
- When a UCS image is to be loaded, it passes control to routine IGG08101.
- If an FCB load is required but a UCS load is not, routine IGG08103 is called.
- If the device is a 3800 printer, control is given to executor IGG08110.
- If the caller's request is invalid, it frees the work area and returns to the caller with a return code in register 15. See *OS/VS2 Data Management Macro Instructions* for a description of the return codes.

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:

- 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.
  2. If the operator replies with a "u" the current image is 'force loaded.'

- If an error occurs during BLDL processing, it returns control to the caller (for non-FCB printers), with a nonzero return code in register 15. 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 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.
- 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 non-zero 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 non-zero return code in register 15.
- 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 the FCB is not to be loaded, FREEMAIN macro instructions release the work areas, a DELETE macro instruction releases the message CSECT, and control is returned to the user's program.

If an FCB image is specified, routine IGG08103 is called; otherwise, control is given to the user's program.

SETPRT Routine IGG08103: Routine IGG08103 locates the FCB image in the problem program's DCB exit list or in SYS1.IMAGELIB and loads it into virtual storage. It is entered from IGC0008A, IGG08101, or IGG08102.

The routine operates as follows:

- 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.
- If an error occurs during BLDL processing, it returns control to the caller, with a non-zero return code in register 15.

- If the image is not found in the library, the routine requests the operator to specify an alternate FCB image.

  1. If the operator cancels the job step, it returns to the caller, with a non-zero return code in register 15.

  2. If the operator replies with a "u", the current image is 'force loaded.'

- Before returning to the caller when an error condition exists, it issues DELETE macro instructions for the message CSECT and for the FCB image, if it was loaded. It also frees the work areas and issues a CLOSE macro instruction for SYS1.IMAGELIB.

- If the FCB image is specified in the problem program or successfully loaded into virtual storage from SYS1.IMAGELIB, routine 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:

- It loads the forms control buffer with the specified image.

- If VERIFY is specified, the image is printed for visual verification.

- If an I/O error occurs during verification, control is returned to the caller with a non-zero 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 problem program.

SETDEV Routine IGG08108: IBM 3890 Document Processor or IBM 3886 Optical Character Reader 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: Executor IGG08110 is entered from executor IGC0008A when the UCB device type indicates that SETPRT processing is being done for a 3800 printer.

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 printer 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.

- 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 printer SETPRT executor specified in the SETPRT path table.

SETPRT Executor IGG08111: 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.
- It formats the translate tables, using the character arrangement tables.
- It uses the EXCP macro to load the 3800 printer'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 printer'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 printer SETPRT executor specified in the SETPRT path table.

SETPRT Executor IGG08112: 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 printer'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.
- 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 printer with the copy modification record.
- It issues a DELETE macro to free the storage used by the loaded copy modification module.

- 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 printer SETPRT executor specified in the SETPRT path table.

SETPRT Executor IGG08113: 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 printer.
- If FCB image verification is requested, IGG08113 formats and prints on the 3800 printer 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 printer SETPRT executor specified in the SETPRT path table.

SETPRT Executor IGG08114: Executor IGG08114 is entered when the UCB extension indicates that the 3800 printer 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: Executor IGG08115 is the last module executed for 3800 SETPRT processing.

The executor operates as follows:

- If no errors were detected by previous 3800 printer SETPRT executors, IGG08115 initializes the 3800 printer with: the starting copy number, the total number of copies to be printed, and the total number of copies to contain a forms overlay image.
- It resets the translate table index in the 3800 printer 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 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 printer. 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. Since the printer is in a not-ready state, no message is written to the 3800.
- 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.
  1. 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.
  2. 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 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:

  1. It deletes the message CSECT (IGGMSG01).
  2. It restores the IOB.
  3. It saves return and reason codes in the SVRB.
  4. 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
  5. It places the return and reason codes in registers 15 and 0, and returns to the caller.
  6. 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 Executor 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 0s, 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.

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 where the user would have to relate a program check in an unfamiliar system routine with an error in his 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 his 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 his 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 his 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 clean-up 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 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.

  Clean-Up

  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 his 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 dis-establish 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 dis-establish 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 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 two-byte code into a three-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 zero.
- 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 eight bytes, followed by two blank bytes.
- It tests for an end of data condition as the buffer is being filled.

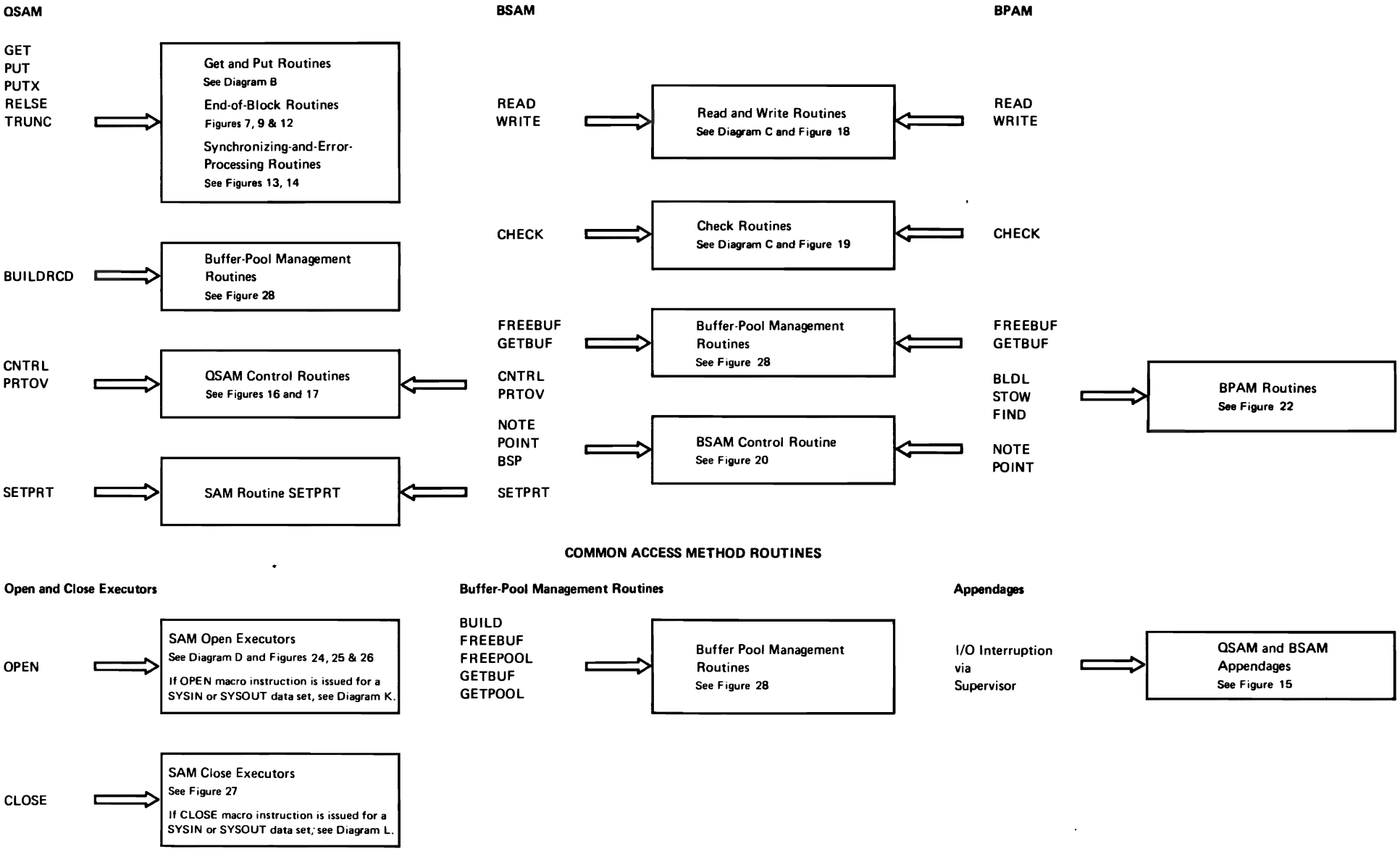
  When the end of data is reached, it restores the registers and exits to EDIT with a return code of four, which requests EDIT to print the contents of the output buffer and obtain the next input trace record.

  If it is not an end of data, it passes control to EDIT with a return code of zero 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.

PROGRAM ORGANIZATION AND FLOW OF CONTROL

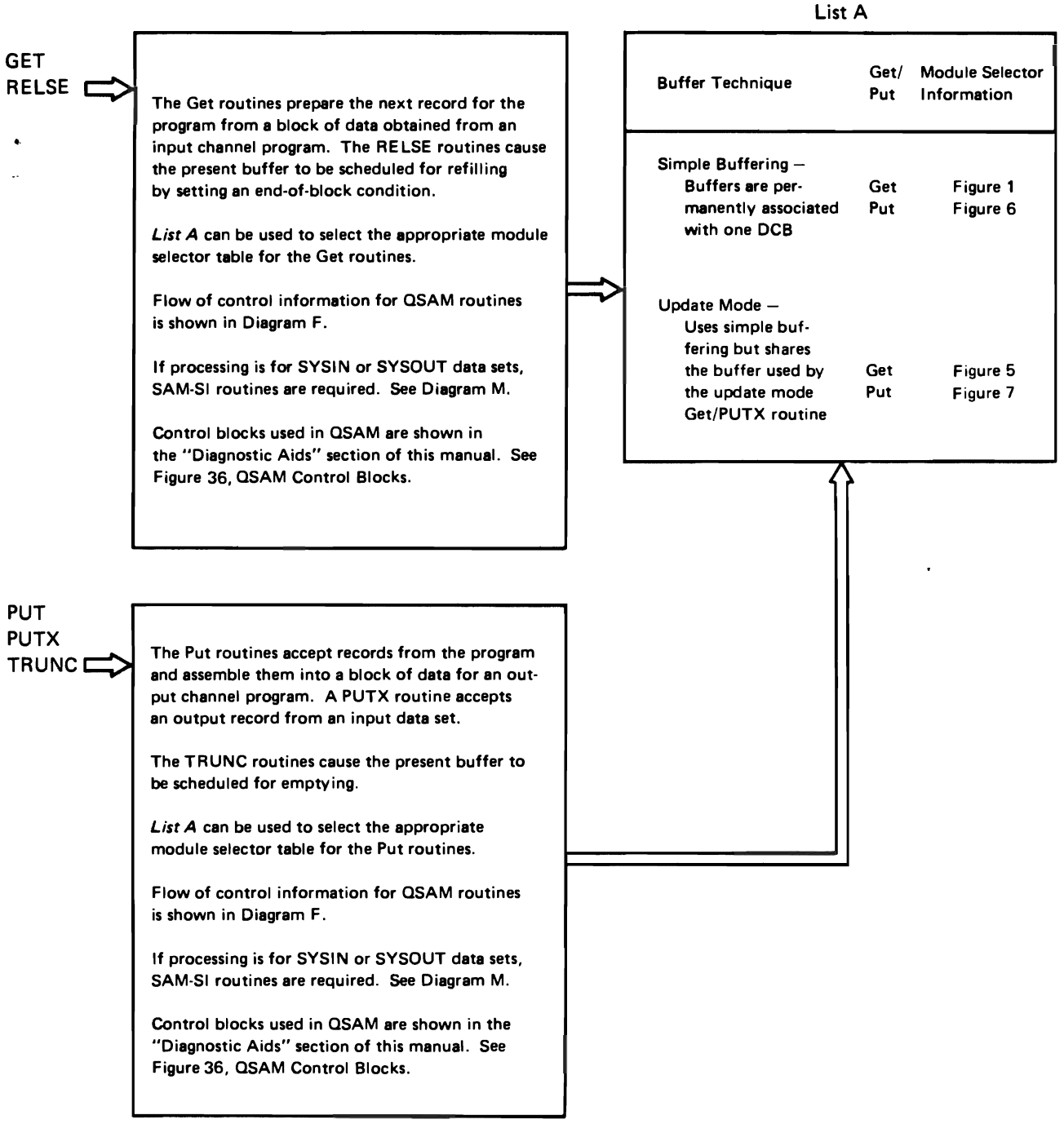
Sequential Access Methods–Overview

DIAGRAM A



QSAM Get and Put Routines

DIAGRAM B



BSAM/BPAM Read/Write and Check Routines

DIAGRAM C

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 18.

For flow of control information for BSAM/BPAM routines, see Figure 22 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 37, 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 19.

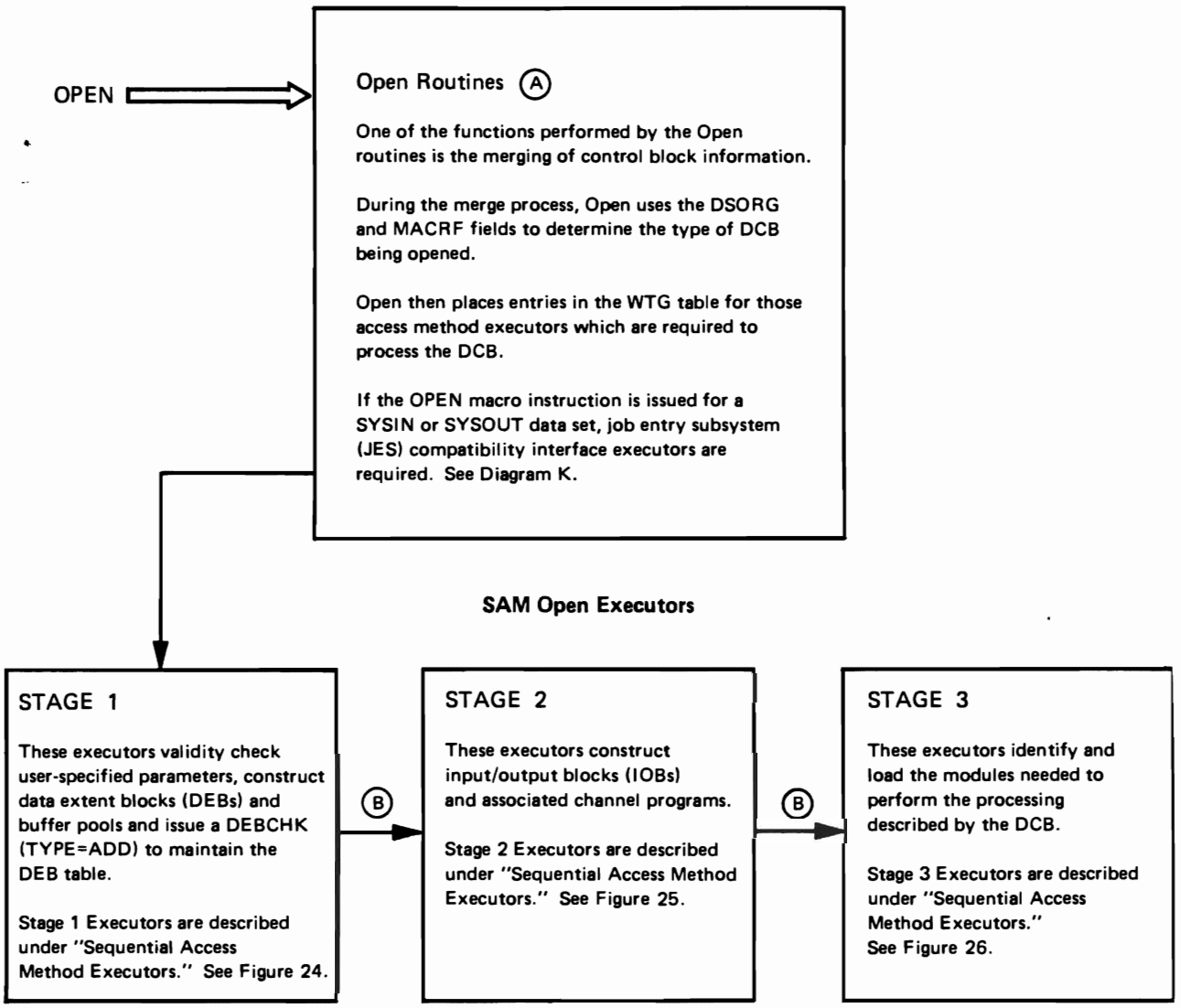
For flow of control information for BSAM/BPAM routines, see Figure 22 and Diagram G.

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 37, BSAM Control Blocks

Sequential Access Method Open Executors

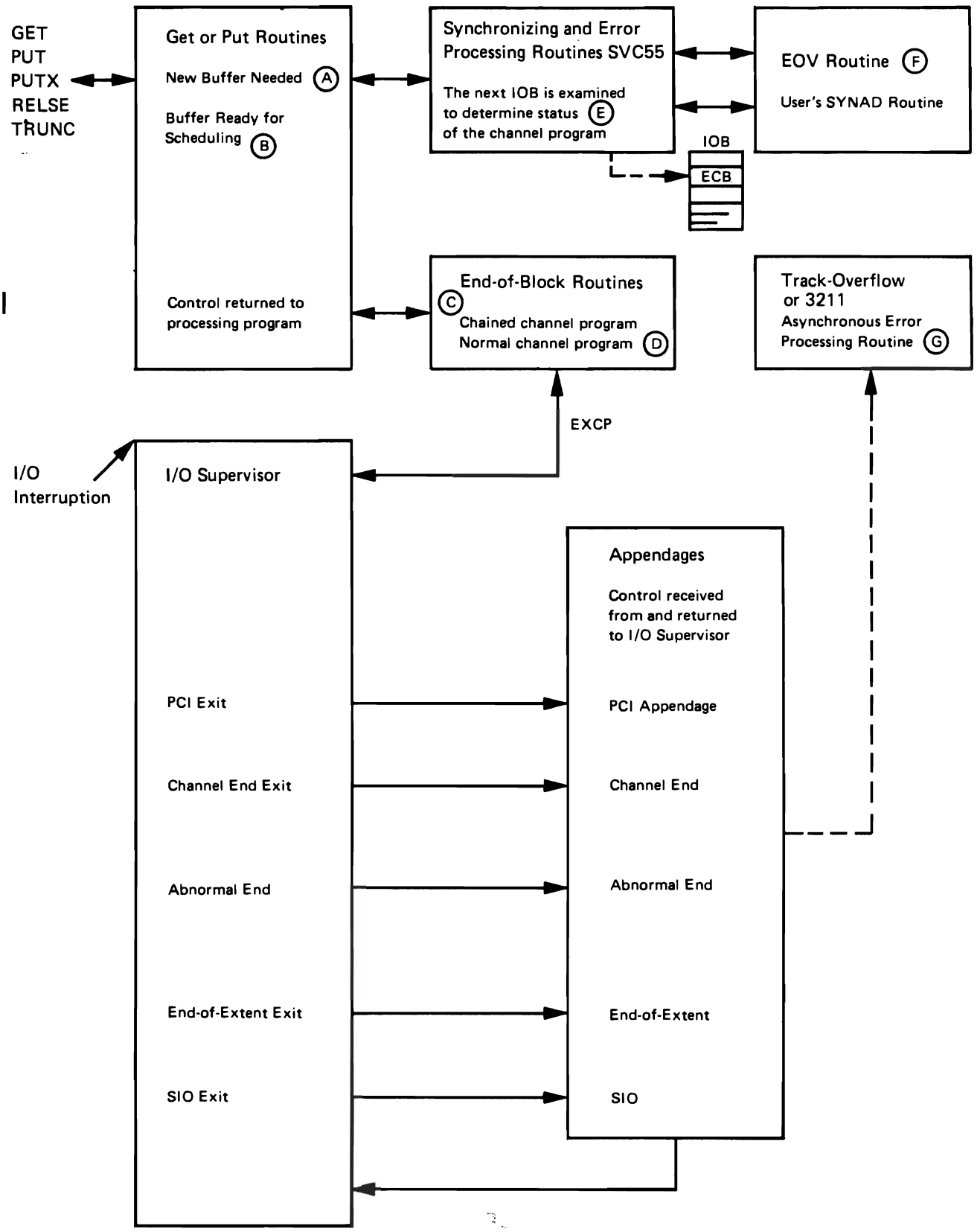
DIAGRAM D



(A)	Open routines are described in *OS/VS2 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.

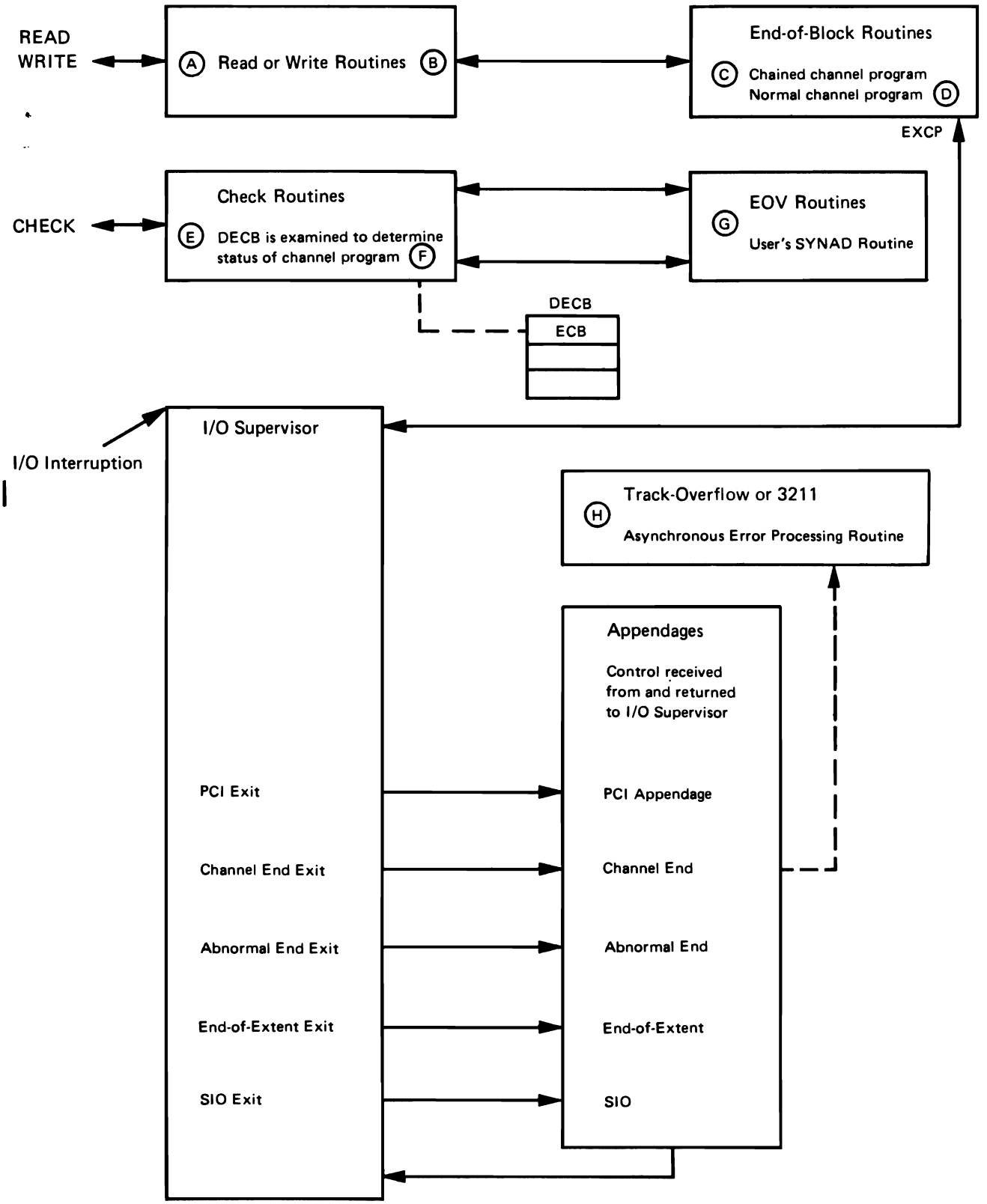
QSAM Flow of Control

DIAGRAM F

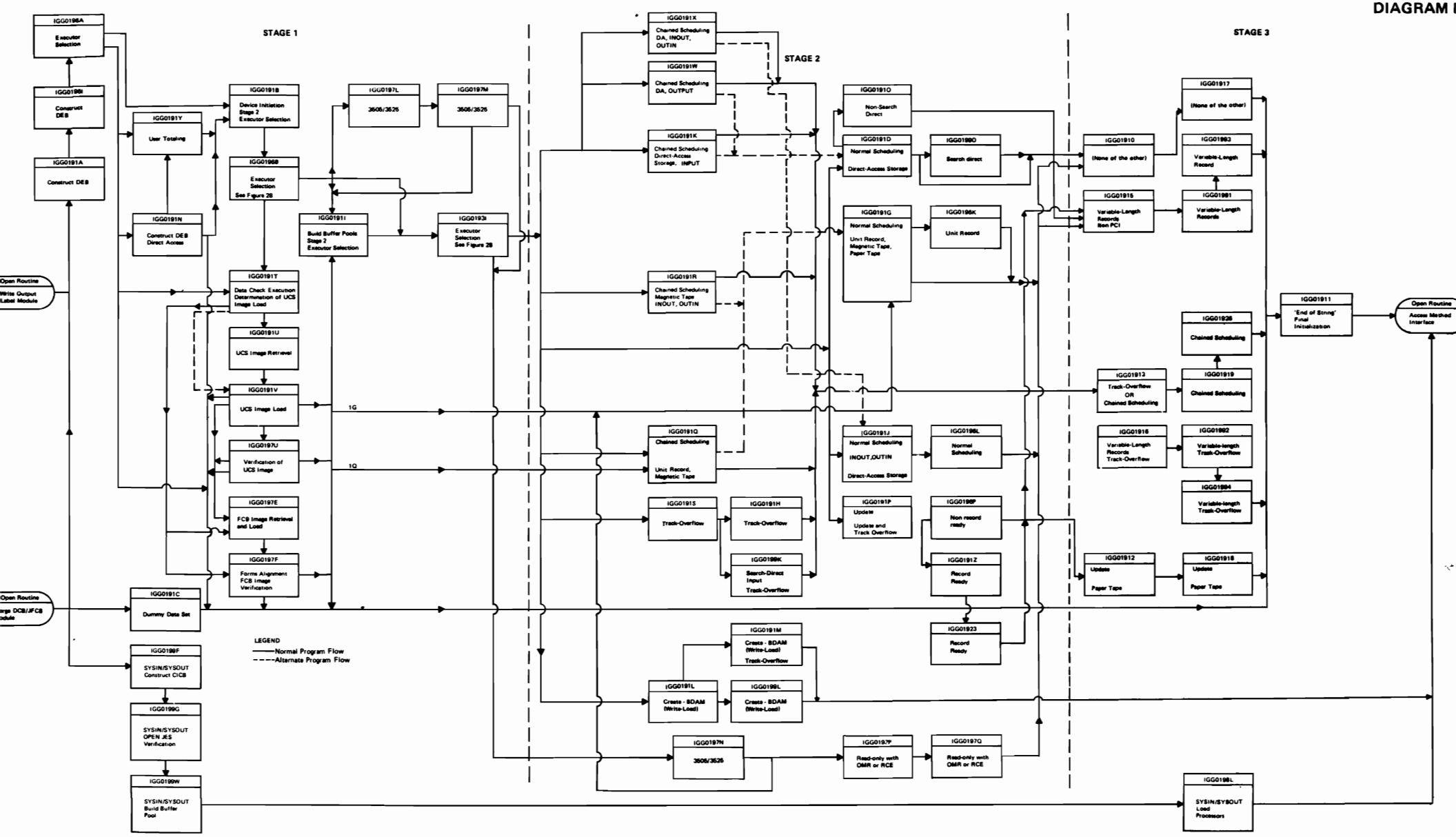


BSAM/BPAM Flow of Control

DIAGRAM G

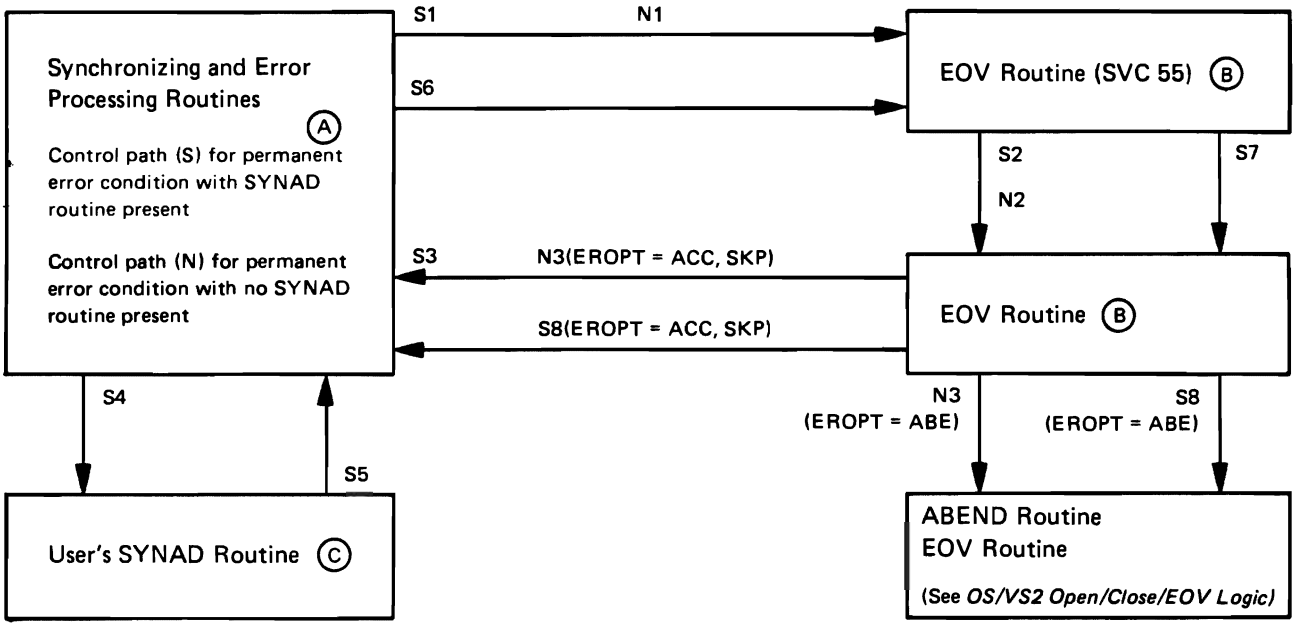


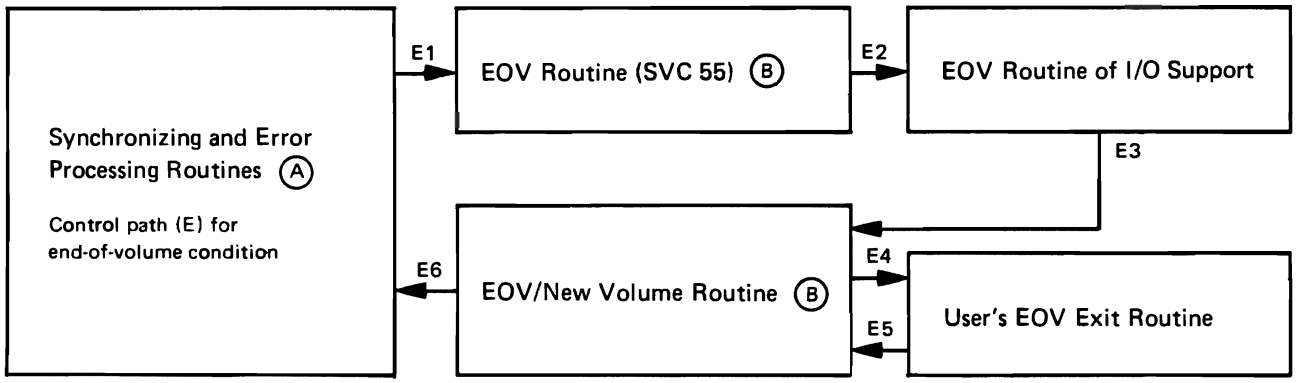
SAM Flow of Control for Open Executors



QSAM Flow of Control with EOV Routines

DIAGRAM H

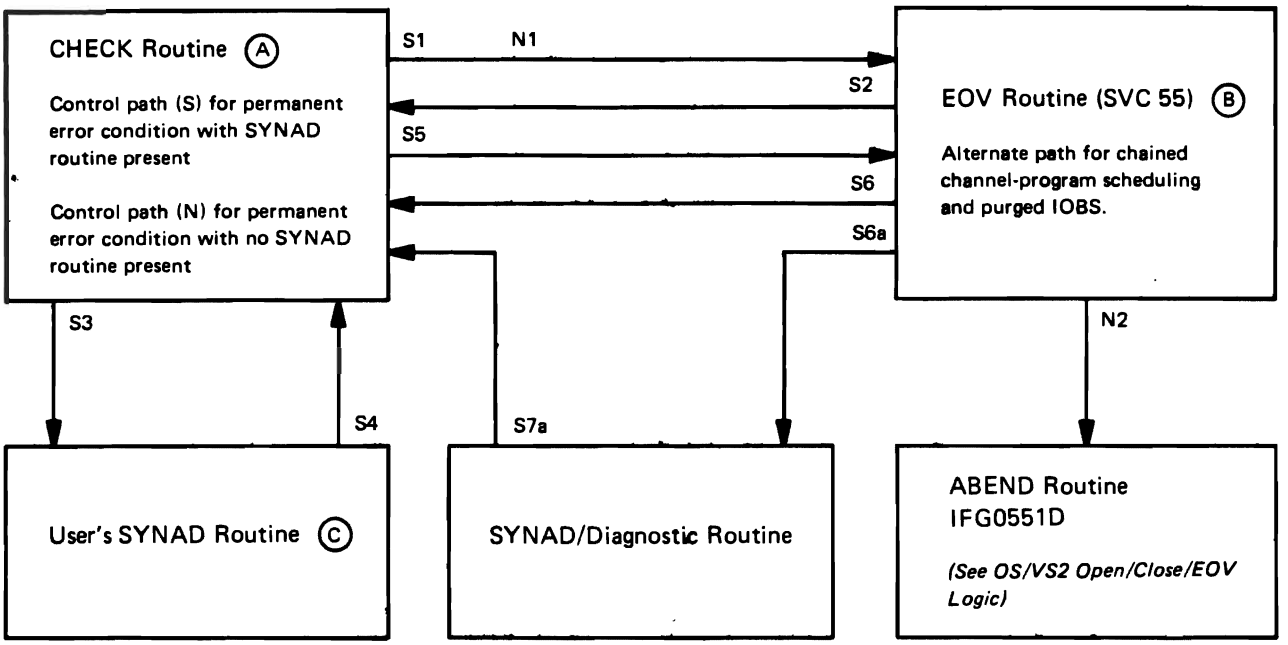


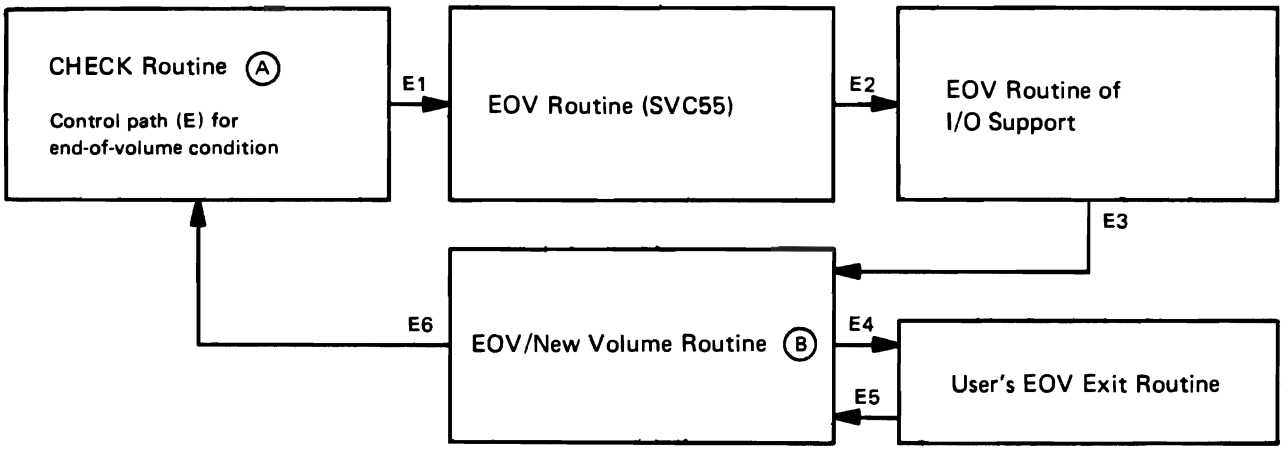


(A)	Descriptive information on these routines is located in "Synchronizing and Error Processing Routines." See module selector information in Figures 13 and 14.
(B)	See *OS/VS2 Open/Close/EOV Logic*.
(C)	The user's SYNAD routine is described in *OS/VS Data Management Services Guide*.

BSAM Flow of Control with EOV Routines

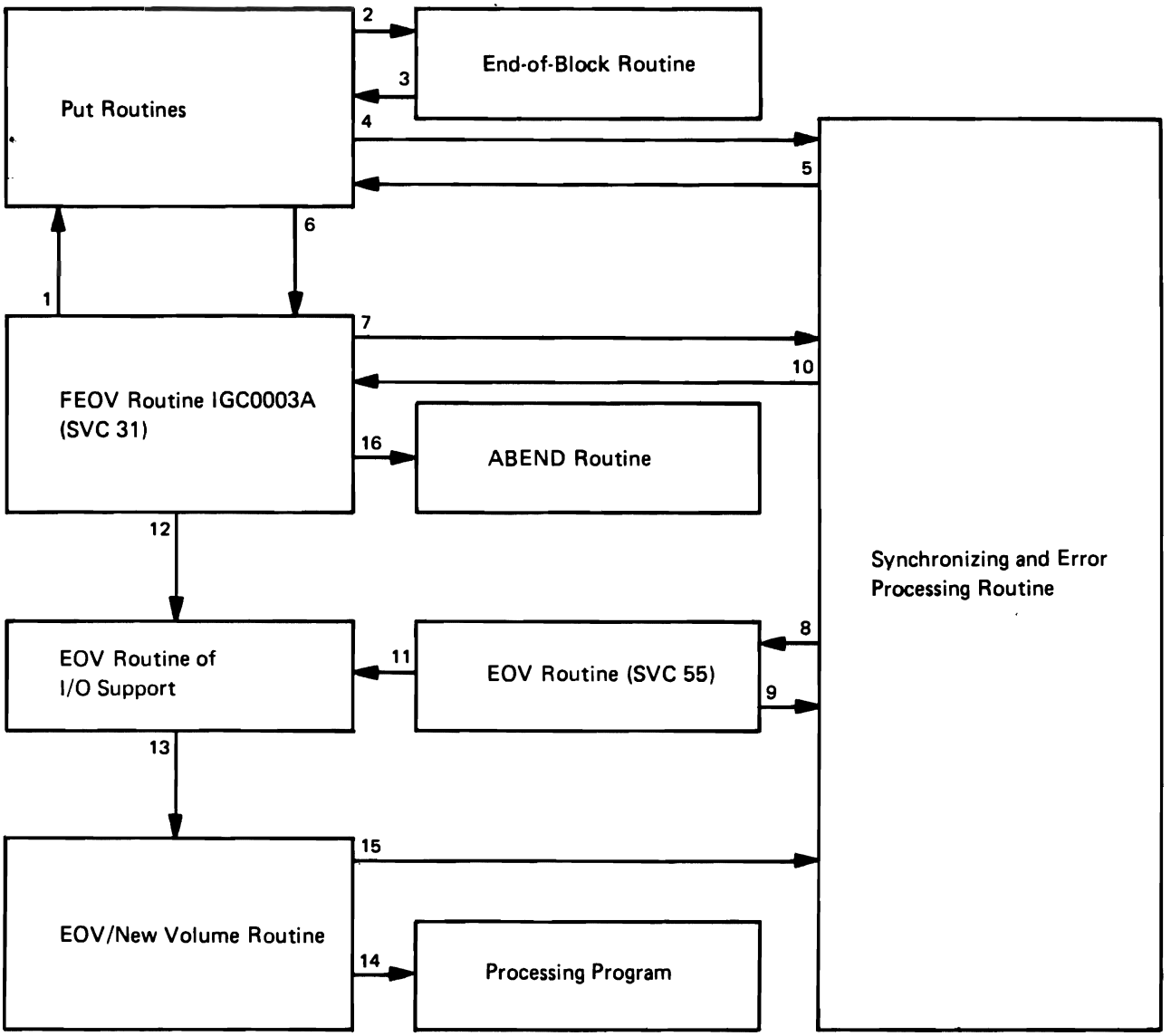
DIAGRAM I





(A)	Descriptive information on the Check routines is located in the "Method of Operation" section under "Basic Sequential Access Method Routines." See Figure 19.
(B)	See *OS/VS2 Open/Close/EOV Logic.*
(C)	The user's SYNAD routine is described in *OS/VS Data Management Services Guide.*

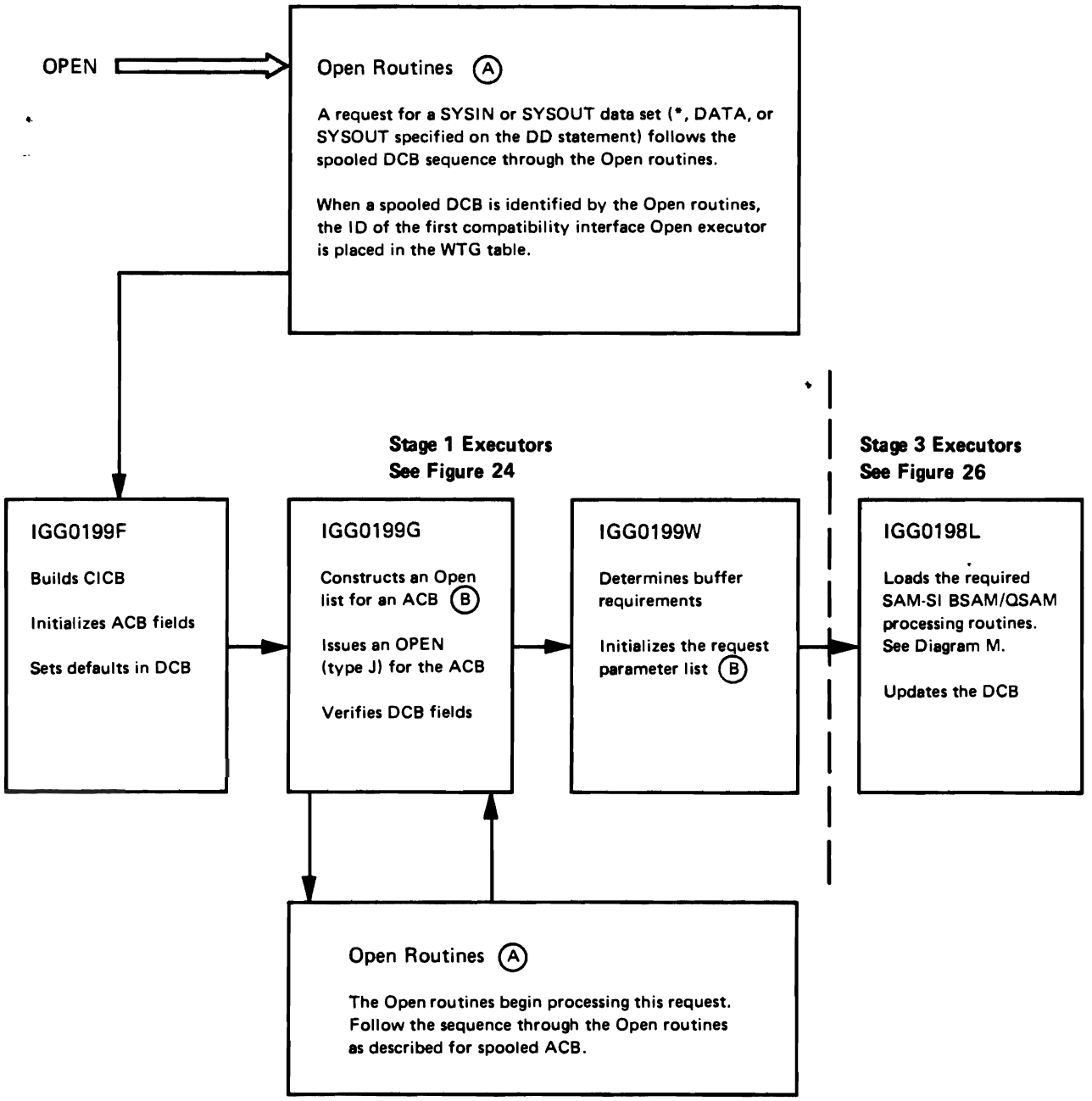
QSAM Operation with FEOV Routine for Output Data Set (IGC0003A) DIAGRAM J



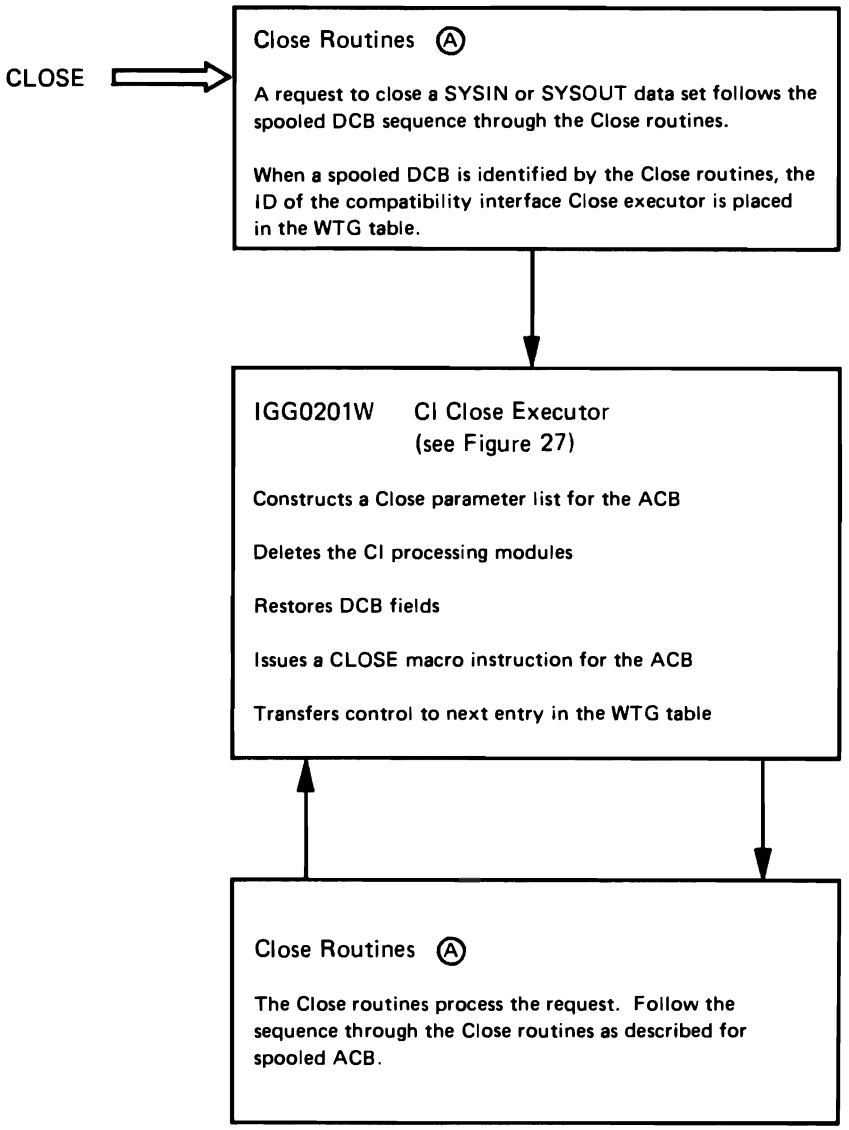
Condition	Sequence of Control
1	1,2,3,6,12,13,14
2	1,2,3,6,7,8,9,10,16
3	1,2,3,6,7,8,11,13,15,10,12,13,14
4	1,2,3,4,5,6,12,13,14
5	1,2,3,4,8,9,10,16
6	1,2,3,4,5,6,7,8,9,10,16
7	1,2,3,4,8,11,13,15,10,12,13,14
8	1,2,3,4,5,6,7,8,11,13,15,10,12,13,14

Open Processing for SAM Subsystem Interface Executors

DIAGRAM K



(A)	Open routines are described in *OS/VS2 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 *OS/VS2 Data Areas.*



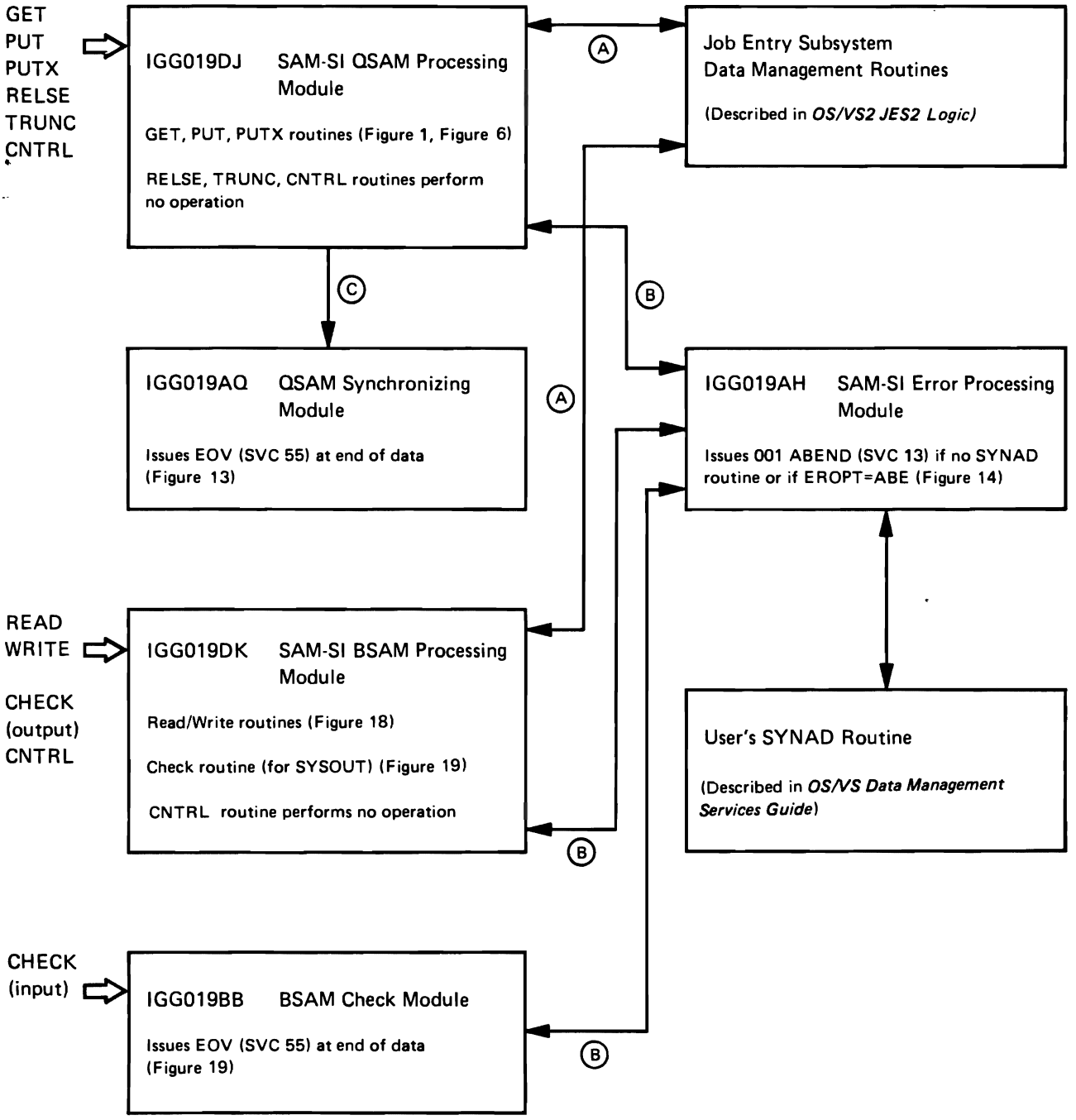
Ⓐ	The Close routines are described in *OS/VS2 Open/Close/EOV Logic.*

SAM Subsystem Interface Flow of Control

Notes for Diagram M

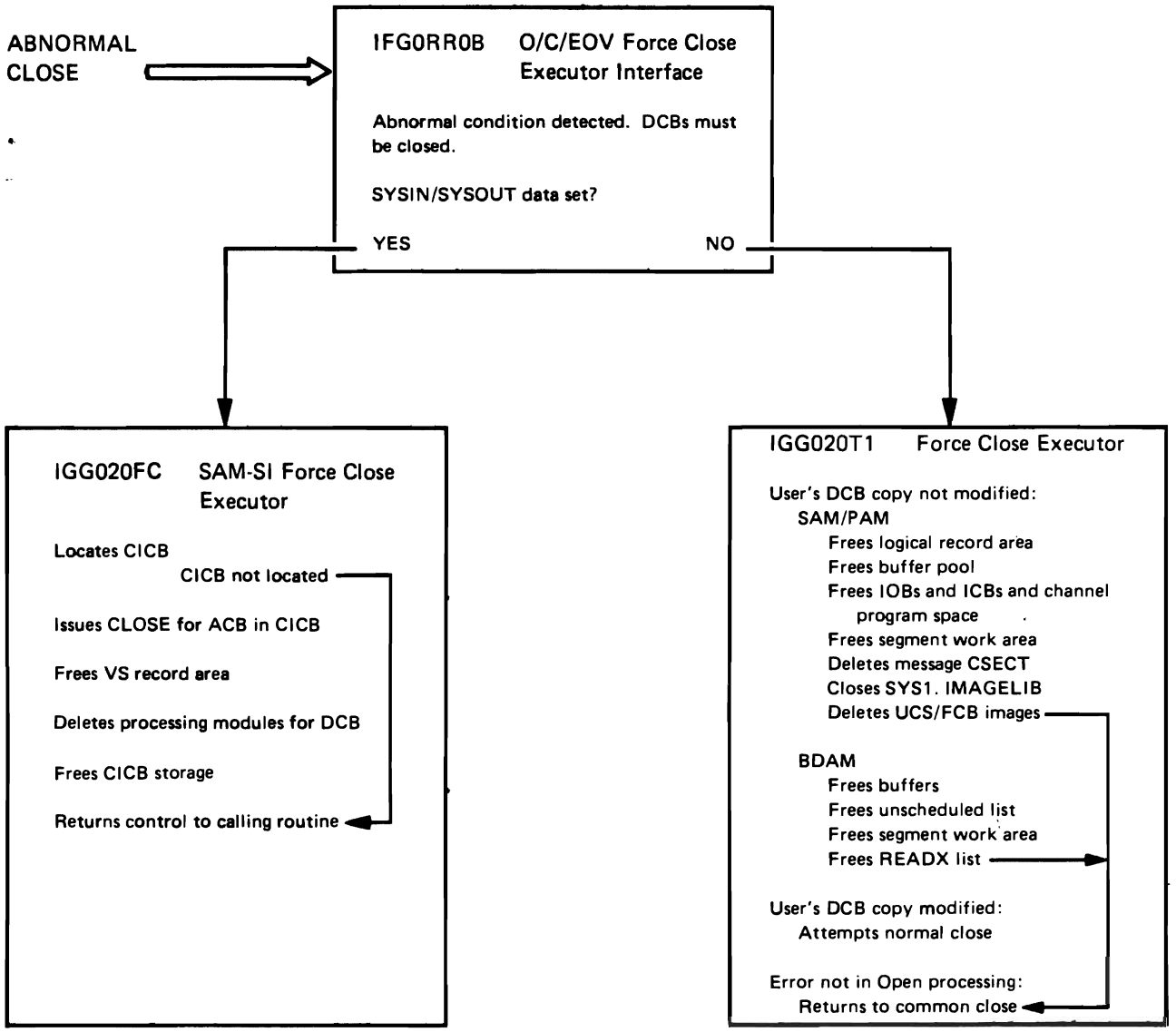
(A)	Processing program requests translation into GET/PUT for the ACB/RPL that gives control to the Job Entry Subsystem.
(B)	Permanent error condition returned by the JES
(C)	End of data condition returned by the JES

SAM Subsystem Interface Flow of Control for SYSIN/SYSOUT Data Sets — DIAGRAM M



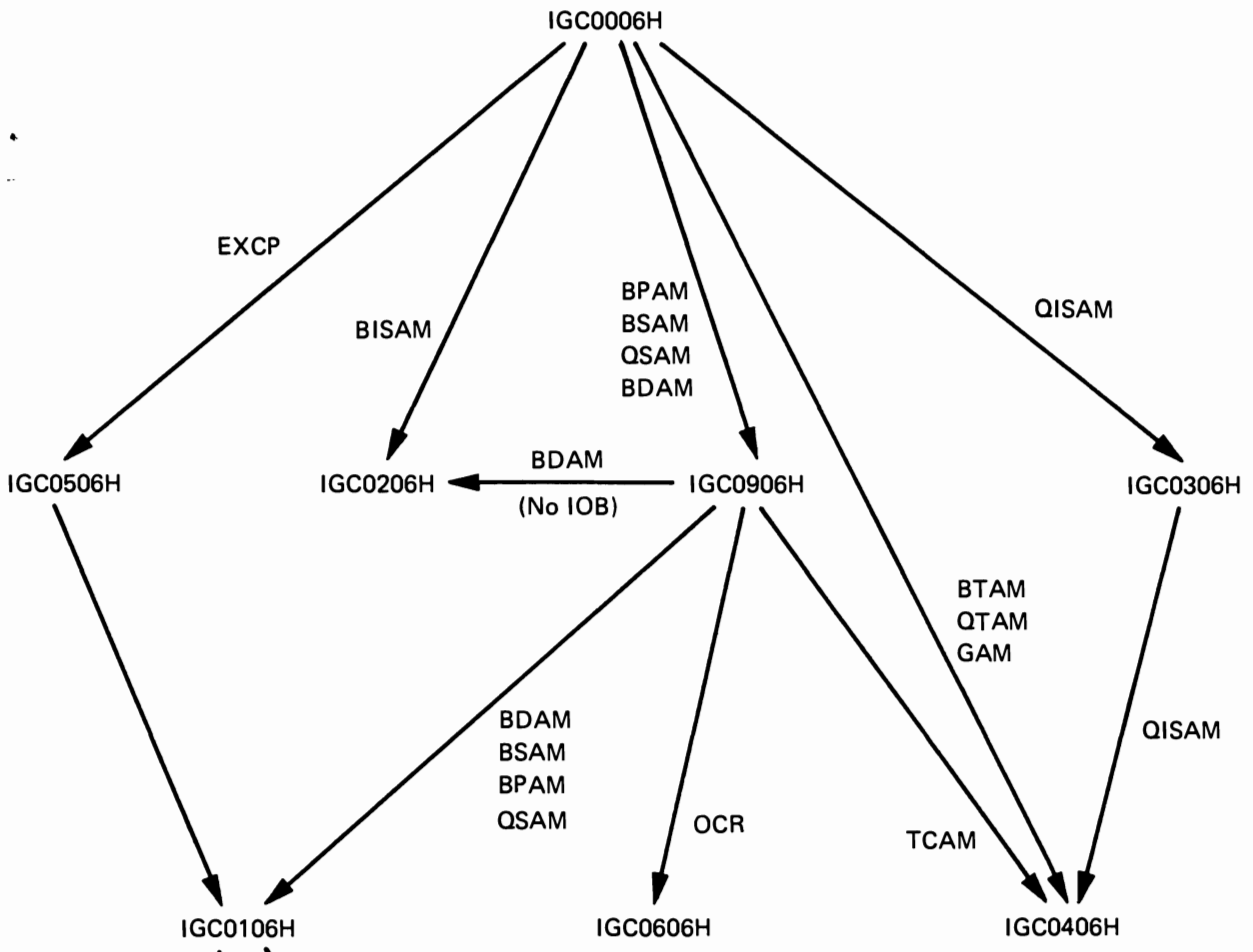
Force Close Processing

DIAGRAM N



SYNADAF Flow of Processing

DIAGRAM O



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

All routines can return to user.

DIRECTORY

Module Name	Module Type	CSECT Name	SVC Entry	Logic Manual Reference	Module Description (Page)
AMDUSRFE	GTRACE Format Appendage	AMDUSRFE			205
IECBBFB1	Build Module	IECBBFB1		Figure 28	179
IECQBFG1	GETPOOL Module	IECQBFG1		Figure 28	178
IFG0559C	Problem Determination Module	IFG0559C			183
IGCT0018	Task Recovery Routine Module	IGCT0018			201
IGCT002D	Task Recovery Routine Module	IGCT002D			202
IGCT002E	Task Recovery Routine Module	IGCT002E			202
IGCT0021	Task Recovery Routine Module	IGCT0021			201
IGCT006H	Task Recovery Routine Module	IGCT006H			202
IGCT0069	Task Recovery Routine Module	IGCT0069			204
IGCT010E	Task Recovery Routine Module	IGCT010E			205
IGCT1081	Task Recovery Routine Module	IGCT1081			204
IGC0002A	STOW Module	IGC0002A	21	Figure 22	188
IGC0002D	DEVTYPE Module	IGC0002D	24		185
IGC0002E	Control Module	IGC0002E	25		186
IGC0006H	SYNADAF Module	IGC0006H	68		192
IGC0006I	Control Module	IGC0006I	69		187
IGC0008A	SETPRT Routine	IGC0008A	81		197
IGC0010C	Translate Routine	IGC0010C	103	See Appendix A	
IGC0010E	IMGLIB Routine	IGC0010E	105		185
IGC0106H	SYNADAF Module	IGC0106H			193
IGC018	Resident Module	IGC018	18	Figure 22	190
		IECPCNVT			191
		IECPRLTV			192
		IEC0SCR1			192
IGC0206H	SYNADAF Module	IGC0206H			194
IGC0306H	SYNADAF Module	IGC0306H			194
IGC0406H	SYNADAF Module	IGC0406H			195
IGC0506H	SYNADAF Module	IGC0506H			195
IGC0606H	SYNADAF Module	IGC0606H			195
IGC0706H	SYNADAF Module	IGC0706H			196
IGC0806H	SYNADAF Module	IGC0806H			196
IGC0906H	SYNADAF Module	IGC0906H			196
IGG019AA	Get Module	IGG019AA		Figure 1	19
IGG019AB	Get Module	IGG019AB		Figure 1	20
IGG019AC	Get Module	IGG019AC		Figure 1	21
IGG019AD	Get Module	IGG019AD		Figure 1	22
IGG019AE	Get Module	IGG019AE		Figure 5	29
IGG019AF	Synchronizing Module	IGG019AF		Figure 13	79
IGG019AG	Get Module	IGG019AG		Figure 1	23
IGG019AH	Error-Processing Module	IGG019AH		Figure 14	84
IGG019AI	Put Module	IGG019AI		Figure 6	43
IGG019AJ	Put Module	IGG019AJ		Figure 6	43
IGG019AK	Put Module	IGG019AK		Figure 6	44
IGG019AL	Put Module	IGG019AL		Figure 6	45
IGG019AM	Get Module	IGG019AM		Figure 1,2	23
IGG019AN	Get Module	IGG019AN		Figure 1,2	24
IGG019AQ	Synchronizing Module	IGG019AQ		Figure 13	81
IGG019AR	Synchronizing Module	IGG019AR		Figure 13	82

Module Name	Module Type	CSECT Name	SVC Entry	Logic Manual Reference	Module Description (Page)
IGG019AT	Get Module	IGG019AT		Figure 1,13	26
IGG019AV	Open Executor	IGG019AV			
IGG019AW	End-of-Extent Appendage	IGG019AW		Figure 15	90
IGG019AX	Save Module	IGG019AX		Figure 7,9,12	75
IGG019BA	Read/Write Module	IGG019BA		Figure 18	114
IGG019BB	Check Module	IGG019BB		Figure 19	123
IGG019BC	Control Module	IGG019BC		Figure 20	128
IGG019BD	Control Module	IGG019BD		Figure 20	130
IGG019BE	Control Module	IGG019BE		Figure 20	131
IGG019BF	Read Module	IGG019BF		Figure 18	116
IGG019BG	Check Module	IGG019BG		Figure 19	125
IGG019BH	Read/Write Module	IGG019BH		Figure 18	117
IGG019BI	Check Module	IGG019BI		Figure 19	126
IGG019BK	Control Module	IGG019BK		Figure 20	132
IGG019BL	Control Module	IGG019BL		Figure 20	133
IGG019BM	End-of-Extent Appendage	IGG019BM		Figure 15	90
IGG019BN	Get Update Module	IGG019BN		Figure 5	37
IGG019BO	Get Module	IGG019BO		Figure 1	27
IGG019BP	Put Module	IGG019BP		Figure 6	46
IGG019BQ	Synchronizing Module	IGG019BQ		Figure 13	82
IGG019BR	Write Module	IGG019BR		Figure 18	119
IGG019BS	Check Module	IGG019BS		Figure 19	126
IGG019BV	Channel End Appendage	IGG019BV		Figure 15	96
IGG019BT	Channel End Appendage	IGG019BT		Figure 15	95
IGG019BU	Read Module	IGG019BU		Figure 18	119
IGG019B0	BUILDRCD Routine	IGG019B0		Figure 28	181
IGG019CA	Control Module	IGG019CA		Figure 16,21	112
IGG019CB	Control Module	IGG019CB		Figure 16,21	113
IGG019CC	EOB Module	IGG019CC		Figure 7	54
IGG019CD	EOB Module	IGG019CD		Figure 7	56
IGG019CE	EOB Module	IGG019CE		Figure 7	58
IGG019CF	EOB Module	IGG019CF		Figure 7	59
IGG019CG	SIO Appendage	IGG019CG		Figure 15	94
IGG019CH	End-of-Extent Appendage	IGG019CH		Figure 15	91
IGG019CI	Channel-End and Abnormal-End Appendage	IGG019C1		Figure 15	96
IGG019CJ	Channel-End and Abnormal-End Appendage	IGG019CJ		Figure 15	98
IGG019CL	SIO Appendage	IGG019CL		Figure 15	94
IGG019CM	Code Conversion Module	IGG019CM		See Appendix A	
IGG019CN	Code Conversion Module	IGG019CN		See Appendix A	
IGG019CO	Code Conversion	IGG019CO		See Appendix A	
IGG019CP	Code Conversion Module	IGG019CP		See Appendix A	
IGG019CQ	Code Conversion Module	IGG019CQ		See Appendix A	
IGG019CR	Code Conversion Module	IGG019CR		See Appendix A	
IGG019CS	Channel-End Appendage	IGG019CS		Figure 15	100
IGG019CT	EOB Module	IGG019CT		Figure 7,12	60
IGG019CU	Appendage	IGG019CU		Figure 15	106
IGG019CV	EOV Module	IGG019CV		Figure 9	67
IGG019CW	EOB Module	IGG019CW		Figure 9	68
IGG019CX	EOB Module	IGG019CX		Figure 9	69
IGG019CY	EOB Module	IGG019CY		Figure 9	70
IGG019CZ	End-of-Extent Appendage	IGG019CZ		Figure 15	92
IGG019C0	Channel-End and Abnormal-End Appendage	IGG019C0		Figure 15	101
IGG019C1	Asynchronous Error Processing Module	IGG019C1		Figure 14	76
IGG019C2	EOB Module	IGG019C2		Figure 12	75
IGG019C3	Channel-End and Abnormal-End	IGG019C3		Figure 15	109

Module Name	Module Type	CSECT Name	SVC Entry	Logic Manual Reference	Module Description (Page)
	Appendage				
IGG019C4	End-of-Extent Appendage	IGG019CA		Figure 15	92
IGG019DA	Write Module	IGG019DA		Figure 18	119
IGG019DB	Write Module	IGG019DB		Figure 18	120
IGG019DC	Check Module	IGG019DC		Figure 19	127
IGG019DD	Write Module	IGG019DD		Figure 18	121
IGG019DJ	Get Module	IGG019DJ		Figure 1	28
	Put Module			Figure 6	48
IGG019DK	Read/Write Module	IGG019DK		Figure 18	122
	Check Module			Figure 19	127
IGG019EI	Channel-End and Abnormal End Appendage	IGG019EI		Figure 15	102
IGG019EJ	Channel-End and Abnormal-End Appendage	IGG019ES		Figure 15	104
IGG019FA	Control Module	IGG019FA		Figure 16,20	113
IGG019FB	Get Module	IGG019FB		Figure 1	29
IGG019FD	Get Module	IGG019FD		Figure 1	30
IGG019FF	Get Module	IGG019FF		Figure 1	31
IGG019FG	Put Module	IGG019FG		Figure 6	49
IGG019FJ	Put Module	IGG019FJ		Figure 6	51
IGG019FK	EOB Module	IGG019FK		Figure 7	60
IGG019FL	Put Module	IGG019FL		Figure 6	51
IGG019FP	Channel-End Appendage	IGG019FP		Figure 15	105
IGG019FQ	EOB Module	IGG019FQ		Figure 7	60
IGG019FR	Abnormal-End Appendage	IGG019FR		Figure 15	111
IGG019FS	Asynchronous Error-Processing Module	IGG019FS		Figure 14	86
IGG019FU	EOB Module	IGG019FU		Figure 7	62
IGG019JD	Parallel Input Processing Module	IGG019JD			32
IGG019TC	EOB Module	IGG019TC		Figure 7	62
IGG019TD	EOB Module	IGG019TD		Figure 7	63
IGG019TV	EOB Module	IGG019TV		Figure 9	71
IGG019TW	EOB Module	IGG019TW		Figure 9	73
IGG019T2	EOB Module	IGG019T2		Figure 12	76
IGG019T4	TS Put Routine	IGG019T4		See description of IGG0201A.	
				Figure 27	172
IGG019V6	Appendage	IGG019V6		Figure 15	108
IGG0191A	Stage 1 Open Executor	IGG0191A		Figure 24	137
IGG0191B	Stage 1 Open Executor	IGG0191B		Figure 24	139
IGG0191C	Stage 1 Open Executor	IGG0191C		Figure 24	139
IGG0191D	Stage 2 Open Executor	IGG0191D		Figure 25	150
IGG0191G	Stage 2 Open Executor	IGG0191G		Figure 25	151
IGG0191H	Stage 2 Open Executor	IGG0191H		Figure 25	151
IGG0191I	Stage 1 Open Executor	IGG0191I		Figure 24	140
IGG0191J	Stage 2 Open Executor	IGG0191J		Figure 25	152
IGG0191K	Stage 2 Open Executor	IGG0191K		Figure 25	152
IGG0191L	Stage 2 Open Executor	IGG0191L		Figure 25	153
IGG0191M	Stage 2 Open Executor	IGG0191M		Figure 25	154
IGG0191N	Stage 1 Open Executor	IGG0191N		Figure 24	140
IGG0191O	Stage 2 Open Executor	IGG0191O		Figure 25	154
IGG0191P	Stage 2 Open Executor	IGG0191P		Figure 25	154
IGG0191Q	Stage 2 Open Executor	IGG0191Q		Figure 25	155
IGG0191R	Stage 2 Open Executor	IGG0191R		Figure 25	156
IGG0191S	Stage 2 Open Executor	IGG0191S		Figure 25	157
IGG0191T	Stage 1 Open Executor	IGG0191T		Figure 24	141
IGG0191U	Stage 1 Open Executor	IGG0191U		Figure 24	142
IGG0191V	Stage 1 Open Executor	IGG0191V		Figure 24	142

Module Name	Module Type	CSECT Name	SVC Entry	Logic Manual Reference	Module Description (Page)
IGG0191W	Stage 2 Open Executor	IGG0191W		Figure 25	157
IGG0191X	Stage 2 Open Executor	IGG0191X		Figure 25	158
IGG0191Y	Stage 1 Open Executor	IGG0191Y		Figure 24	143
IGG0191Z	Stage 2 Open Executor	IGG0191Z		Figure 25	159
IGG01910	Stage 3 Open Executor	IGG01910		Figure 26	164
IGG01911	Stage 3 Open Executor	IGG01911		Figure 26	164
IGG01912	Stage 3 Open Executor	IGG01912		Figure 26	166
IGG01913	Stage 3 Open Executor	IGG01913		Figure 26	166
IGG01915	Stage 3 Open Executor	IGG01915		Figure 26	167
IGG01916	Stage 3 Open Executor	IGG01916		Figure 26	167
IGG01917	Stage 3 Open Executor	IGG01917		Figure 26	168
IGG01918	Stage 3 Open Executor	IGG01918		Figure 26	168
IGG01919	Stage 3 Open Executor	IGG01919		Figure 26	169
IGG01923	Stage 2 Open Executor	IGG01923		Figure 25	159
IGG01926	Stage 3 Open Executor	IGG01926		Figure 26	169
IGG01931	Stage 1 Open Executor	IGG01931		Figure 24	143
IGG0196A	Stage 1 Open Executor	IGG0196A		Figure 24	144
IGG0196B	Stage 1 Open Executor	IGG0196B		Figure 24	144
IGG0196I	Stage 2 Open Executor	IGG0196I		Figure 24	145
IGG0196K	Stage 2 Open Executor	IGG0196K		Figure 25	159
IGG0196L	Stage 2 Open Executor	IGG0196L		Figure 25	159
IGG0196P	Stage 2 Open Executor	IGG0196P		Figure 25	160
IGG0196Q	Stage 1 Open Executor	IGG0196Q		Figure 24	145
IGG0196R	Stage 1 Open Executor	IGG0196R		Figure 24	145
IGG0196S	Stage 2 TS Module	IGG0196S		See OS/VS TCAM Logic	
IGG0197E	Stage 1 Open Executor	IGG0197E		Figure 24	145
IGG0197F	Stage 1 Open Executor	IGG0197F		Figure 24	145
IGG0197L	Stage 1 Open Executor	IGG0197L		Figure 24	146
IGG0197M	Stage 2 Open Executor	IGG0197M		Figure 24	146
IGG0197N	Stage 2 Open Executor	IGG0197N		Figure 25	160
IGG0197P	Stage 2 Open Executor	IGG0197P		Figure 25	161
IGG0197Q	Stage 2 Open Executor	IGG0197Q		Figure 25	161
IGG0197U	Stage 1 Open Executor	IGG0197U		Figure 24	147
IGG0197V	Stage 1 Open Executor	IGG0197V		Figure 25	161
IGG0198L	Stage 3 Open Executor	IGG0198L		Figure 26	169
IGG0199F	Stage 3 Open Executor	IGG0199F		Figure 24	147
IGG0199G	Stage 3 Open Executor	IGG0199G		Figure 24	147
IGG0199K	Stage 2 Open Executor	IGG0199K		Figure 25	161
IGG0199L	Stage 2 Open Executor	IGG0199L		Figure 25	162
IGG0199O	Stage 3 Open Executor	IGG0199O		Figure 25	164
IGG0199W	Stage 1 Open Executor	IGG0199W		Figure 24	148
IGG01991	Stage 3 Open Executor	IGG01991		Figure 26	170
IGG01992	Stage 3 Open Executor	IGG01992		Figure 26	170
IGG01993	Stage 3 Open Executor	IGG01993		Figure 26	171
IGG01994	Stage 3 Open Executor	IGG01994		Figure 26	171
IGG020FC	SAM-SI Force Close Executor	IGG020FC		Diagram N	177
IGG020T1	SAM/PAM/DAM Force Close Executor	IGG020T1		Diagram N	177
IGG0201A	Close Executor	IGG0201A		Figure 27	172
IGG0201B	Close Executor	IGG0201B		Figure 27	173
IGG0201P	Close Executor	IGG0201P		Figure 27	174
IGG0201R	Close Executor	IGG0201R		Figure 27	174
IGG0201W	Close Executor	IGG0201W		Figure 27	175
IGG0201X	Close Executor	IGG0201X		Figure 27	175
IGG0201Y	Close Executor	IGG0201Y		Figure 27	176
IGG0201Z	Close Executor	IGG0201Z		Figure 27	176
IGG021AB	STOW Module	IGG021AB		Figure 22	190
IGG0210A	STOW Module	IGG0210A		Figure 22	189
IGG08101	SETPRT Routine	IGG08101			198
IGG08102	SETPRT Routine	IGG08102			198
IGG08103	SETPRT Routine	IGG08103			199

Module Name	Module Type	CSECT Name	SVC Entry	Logic Manual Reference	Module Description (Page)
IGG08104	SETPRT Routine	IGG08104			200
IGG08108	SETDEV Routine	IGG08108			200
IGG08110	SETPRT Executor	IGG08110		Figure 34.1	200
IGG08111	SETPRT Executor	IGG08111		Figure 34.1	200
IGG08112	SETPRT Executor	IGG08112		Figure 34.1	200.1
IGG08113	SETPRT Executor	IGG08113		Figure 34.1	200.1
IGG08114	SETPRT Executor	IGG08114		Figure 34.1	200.1
IGG08115	SETPRT Executor	IGG08115		Figure 34.1	200.2
IGG08116	SETPRT Executor	IGG08116		Figure 34.1	200
IGG08117	SETPRT Executor	IGG08117		Figure 34.1	200

The modules of OPEN, CLOSE, STOW and SYNADAF are link edited into SYS1.LPALIB during system generation by macro, SCIEC4DI, according to the following table. Each CSECT is a separate module in the distribution library and in microfiche. For Open and Close each CSECT name is also an alias for the load module.

Load Module	CSECTs
IGG0191A	IGG0191A, IGG0191B, IGG0191I, IGG0191N, IGG0191T, IGG0191U, IGG0191V, IGG0191Y, IGG0193I, IGG0196A, IGG0196B, IGG0196I, IGG0197E, IGG0197F, IGG0197U
IGG01911	IGG0191C, IGG0191H, IGG0191K, IGG0191Q, IGG0191R, IGG0191S, IGG0191W, IGG0191X, IGG01911, IGG01916, IGG01918, IGG01919, IGG01926, IGG0199K, IGG01992, IGG01994
IGG0191D	IGG0191D, IGG0191G, IGG0191J, IGG0191O, IGG0191P, IGG0191Z, IGG01910, IGG01912, IGG01915, IGG01917, IGG01923, IGG0196K, IGG0196L, IGG0196P, IGG0199O, IGG01991, IGG01993
IGG0201Z	IGG0201A, IGG0201B, IGG0201X, IGG0201Y, IGG0201Z
IGC0002A	IGC0002A, IGG021AB, IGG0210A
IGC0006H	IGC0006H, IGC01016H, IGC0206H, IGC0306H, IGC0406H, IGC0506H, IGC0606H, IGC0706H, IGC0806H, IGG0906H

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	Length	Name	Description
Index			
0(0)	2	MSGINDLN	Length of index
2(2)	2	MSGINDOF	Offset to message entry for first message
4(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			
0(0)	1	MSGOFF	Offset to message from beginning of entry
1(1)	1	MSGLNG	Length of message -1 (to allow use in execution of a move)
2(2)	1	MSGOFF1	Offset to first variable filled in by the routine that extracts the message and causes it to be printed
3(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			
0(0)	1	MSGLNGF	Length of message text
1(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(0)	1	SPWOPTSA	Save area for the opcode byte
1(1)	1	SPWPARM1	Saved reply flags
2(2)	1	SPWFLG1	Reply flags:
	1...	SPWVRFCB	Verify FCB
	.xxx xxx.		Reserved
	1	SPWALIGN	Align forms
3(3)	1	SPWFLG2	Module entry indicator
4(4)	1	SPWFLG3	Flags:
	...1	SPW120RQ	Message IEC120 is required
	xxx. xxxx		Reserved
5(5)	1	SPWFLG4	Flag byte:
	1...	SPWFCBDE	FCB image is loaded and must be deleted
	.1..	SPWECPAM	EXCP DCB with access method section is present
	..xx xxxx		Reserved
6(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(7)	1	Reserved	

WTOR Prefix, Message Section, and Reply Area—in User Key

Offset	Length	Name	Description
8(8)	8	SPWMSGHD	Header section for message area
8(8)	4	SPWRPLYA	Data about the reply area
8(8)	1	SPWMSGLB	Length of the reply area
9(9)	3	SPWRPLYB	Address of the reply area
12(C)	4	SPWECBPA	Address of the reply ECB
16(10)	80	SPWMSGAR	SPW message area
16(10)	64	SPWMSGTX	Message text
80(50)	16	SPWREPLY	Operator reply
80(50)	1		Reserved
81(51)	15	SPWFCBOR SPWREPC1	Start of operator reply
96(60)	4	SPWRPECB	Reply ECB for WTOR
100(64)	4	SPWREPID	UCS/FCB ID supplied by the operator reply
104(68)	8	SPWFCBIM	Name of the FCB image
104(68)	8	SPWUCSIM	Name of the UCS image
108(6C)	4	SPWUCS2H	Last 4 bytes of UCS name
108(6C)	4	SPWFCB2H	Last 4 bytes of FCB name

IOB For EXCP Users and OPEN—in User Key

Offset	Length	Name	Description
112(70)	40	SPWIOB	IOB area
Information saved from user's IOB			
152(98)	4	SPWFLGSV	IOB first word save area
156(9C)	4	SPWTRSV	IOBSTART save area

Channel Program Area—in User Key

Offset	Length	Name	Description
160(A0)	8	SPWCCW1	First CCW
168(A8)	8	SPWCCW2	Second CCW
176(B0)	8	SPWCCW3	Third CCW

Work Area For Unpacking Line Numbers—in User Key

Offset	Length	Name	Description
184(B8)	4	SPWUNPKA	Unpack area
188(BC)	2		Unused bytes
190(BE)	2	SPWLNENO	Used by CONVERT instruction

General Work Area—in User Key

Offset	Length	Name	Description
192(C0)	4	SPWFFSB	Sense bytes 0-3 from the 3800 printer
196(C4)	4	SPWMSGID	Message ID for DOM
200(C8)	64	SPWUBLDL	BLDL parameter list for user image library
204(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(0)	64	SPWBLDLA	BLDL list area
0(0)	4	SPWBLDLC	Count and length fields
4(4)	8	SPWDDNAM SPWBLNAM	DD name for SYSOUT request BLDL name field
8(8)	4	SPWBLFCB	Load module ID for BLDL
12(C)	52		Reserved
64(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(44)	8	SPWMSGHS	Header section for message area
		SPWMLIST	Error message parameter list for IGG08116
76(4C)	100	SPWMSGA	SPW message area
		SPWMTXT	Message buffer for IGG08116
176(B0)	4	SPWMRECB	Reply ECB for WTOR
180(B4)	8	SPWIMAGE	Name of the image requested

3800 Printer Subsystem Area for the SETPRT Work Area—Key 5

Offset	Length	Name	Description
188(BC)	4	SPWSPRB	Address of the SVRB extended save area
192(C0)	4	SPWSOADD	Address of the SYSOUT work area for IGG08117
196(C4)	2	SPWSOLEN	Length of the SYSOUT work area
200(C8)	4	SPWTWAD1	Address of the translate table work area
204(CC)	1	SPWTWSP1	Translate table work area subpool number
205(CD)	3	SPWTWLN1	Translate table work area length
208(D0)	1	SPWFLAG1	Flag byte 1:
	1...	SPWT3800	The SETPRT is for a 3800
	.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 section in message area
	.... ..1.	SPWEFCBP	End of FCB verify printout
	1	SPWM128L	Message IEC128D in message area
209(D1)	1	SPWFLAG2	Flag byte 2:
	1...	SPWRVMSG	Reissue verify message to 3800
	.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(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
	..xx		Unused
	.... 1...	SPWGRAF0	WCGM 0 has been GCM modified
	.... .1..	SPWGRAF1	WCGM 1 has been GCM modified
	.... ..1.	SPWGRAF2	WCGM 2 has been GCM modified
	1	SPWGRAF3	WCGM 3 has been GCM modified
211(D3)	1		Unused
212(D4)	4	SPWRSNCD	Reason code
212(D4)	1	SPWRSN0	Byte 0 of the reason code
213(D5)	1	SPWRSN1	Byte 1 of the reason code
214(D6)	1	SPWRSN2	Byte 2 of the reason code
215(D7)	1	SPWRSN3	Byte 3 of the reason code
		SPWREASN	

Offset	Length	Name	Description
216(D8)	4	SPWRETCD	Return code
216(D8)	1	SPWRET0	Byte 0 of the return code
217(D9)	1	SPWRET1	Byte 1 of the return code
218(DA)	1	SPWRET2	Byte 2 of the return code
219(DB)	1	SPWRET3	Byte 3 of the return code
220(DC)	4	SPWIOBST	Address of the IOB standard section
224(E0)	2	SPWLNCNT	FCB image line counter
226(E2)	2	SPWFCBIL	Length of the FCB image
228(E4)	4	SPWCAVTA	Address of the caller's AVT
232(E8)	2	SPWWKBTS	Work bytes used to test flags
234(EA)	1	SPWI	Total number of translate tables
235(EB)	1	SPWJ	Translate table index
236(EC)	1	SPWK	Index of the CGM in the translate tables
237(ED)	1	SPWL	Work index
238(EE)	1	SPWM	Index in the CGM record
239(EF)	1	SPWN	Index in the CGM record
240(F0)	2	SPWP	Translate table position index
242(F2)	1	SPWMAX	Number of CGMs installed on the printer
243(F3)	1	SPWTCBKY	TCB key
244(F4)	4	SPWWCGMS SPWIOREC	Load WCGM data Execute control data
248(F8)	4	SPWCGMID	Character set IDs
252(FC)	20	SPWMEXIT(n)	Five 4-byte areas that contain the SETPRT modules exit list addresses
272(110)	1	SPWMEIND	Index for module exit list
273(111)	1	SPWERIND	Error index for module exit list
274(112)	1	SPWRXIND	Retransmit index for module exit list
275(113)	1		Unused
276(114)	1	SPWPLENG	Length of the SETPRT parameter list
278(116)	2		Unused
280(118)	4	SPWADDCB	Address of the caller's DCB
284(11C)	4	SPWADDEB	Address of the caller's DEB
288(120)	4	SPWADIOB	Address of the IOB prefix section
292(124)	4	SPWADUCB	Address of the UCB
296(128)	1	SPWCFHIT	FCB half-inch counter
297(129)	1	SPWCFB	FCB current index for verify
298(12A)	2		Unused
300(12C)	4		Unused
304(130)	12		Unused
316(13C)	60	SPWSAVE	SETPRT register save area
376(178)	72	SPWRSAVE	Compiler register save area
448(1C0)	8	SWPFLINU	FCB image line number
456(1C8)	56	SPWSPP	Copy of the user SPP to key 5

Offset	Length	Name	Description
512(200)	1	SPWUKSN	Subpool number of work area in user key
513(201)	3	SPWUKLTH	Length of user key work area
516(204)	1	SPWK5SN	Subpool number of work area in key 5
517(205)	3	SPWK5LTH	Length of key 5 work area
520(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(0)	2	SPWMCLEN	Total length of area
2(2)	2	SPWRSV02	Reserved
4(4)	2	SPWRSV04	Reserved
6(6)	2	SPWTXTL	Length of text returned
8(8)	2	SPWRSV08	Reserved
10(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(0)	4	SPRMSG	Address of the message CSECT
4(4)	4	SPRIDCBA	Address of the image library data set's DCB
8(8)	4	SPREXIT	Exit prolog
		SPRREG13	Save area for register 13
12(C)	1	SPRKEY	User key
13(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(E)	1	SPRNIOBS	Number of IOBs
15(F)	1		Unused
16(10)	4	SPRBLDLA	Address of the BLDL list
20(14)	4	SPRIOBSV	Address of the IOB altered by IGC0008A
24(18)	4	SPRDCBBG	Beginning address of the DCB
28(1C)	4	SPRGETMN	Address of the GTRACE buffer
32(20)	16	SPRELIST	ESTAE list
32(20)	8	SPRTRACE	GTRACE list

3800 Printing Subsystem Translate Table Entry—in key 5

Offset	Length	Name	Description
0(0)	288	SPWTT	Translate table, pointed to by SPWTWAD1 (in the SETPRT work area)
0(0)	8	SPWTTHDR	Translate table header
0(0)	4	SPWTTID	Translate table ID
4(4)	2		Reserved
6(6)	2		Length of character arrangement table
8(8)	280	SPWXLAT	Translate table and trailer
8(8)	256	SPWTRANS	256-byte translate table
264(108)	24	SPWTRAIL	Trailer
264(108)	8	SPWTRL1	Four 2-byte entries for character set identifier and loading order
272(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(0)	1	SPWFCBIE	FCB byte for 3800
	00..		Reserved—set to zeros
	..nn	SPWFCBLP	Lines-per-inch bits
	..00		6 lines per inch
	..01		8 lines per inch
	..11		12 lines per inch
	.... nnnn		Channel number 1 to 12, in hexadecimal code

Buffer Pool Control Block—IGGBCB

Offset	Length	Name	Description
0(0)	4	BCBBUFPT	Address of the first buffer (same as BCBBUFAD below)
0(0)	1		Filler
1(1)	3	BCBBUFAD	Address of first buffer
4(4)	1	BCBFLGS	Flag byte
	1...	BCBLRI	Logical record interface present
	.1..	BCBEXTND	BUFCB extended area present
5(5)	1	BCBBUFNO	Number of buffers
6(6)	2	BCBBUFSZ	Size of each buffer
8(8)	4	BCBLRIAR	Address of logical record area (same as BCBLRIAD below)
8(8)	1		Filler
9(9)	3	BCBLRIAD	Address of logical record area
12(C)	4	BCBPAD	Padding for doubleword alignment
	.... 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		
0(0)	8	LRILOC	LRI location
0(0)	4	LRILGTH	Length of LRI area—LRECL+32 (same as LRILNGTH below)
0(0)	1	LRIFLG1	Flags
	1...	LRIEOD	End-of-data reached
	.1..	LRICOB	COBOL data set
	..1.	LRIEOB	EOD after first end-of-block
1(1)	3	LRILNGTH	Length of LRI Area—LRECL+32
4(4)	1	LRIFLAG2	Flags

Offset	Length	Name	Description
	.1..	LRIRELSE	Release issued
	...1	LRISEG	Segmenting is in process
	.... .1..	LRINTSPN	Non-spanned record
	1	LRIASSEM	Assembling is in process
5(5)	1	LRIINDEX	Index to beginning of data
6(6)	2	LRIPOS	Position of record in block
8(8)	8	LRITRKAD	Track address of beginning record segment
8(8)	5	LRIMBBCC	MBBCC of track address (not used if DCB is for output)
13(D)	3	LRIRECAD	Record address when record to be written requires segmentation
16(10)	4	LRINIOB	Next IOB address (same as LRINXIOB below)
16(10)	1		Filler
17(11)	3	LRINXIOB	Next IOB address
20(14)	2	LRICOUNT	Count field of number of bytes moved
22(16)	2		Filler
24(18)	8	LRIALIGN	Floating alignment area
24(18)	1	LRIDATA	Data

Parameter List—IGGPARML

This DSECT expands the parameter list passed to the open/close executors from common open/close.

Offset	Length	Name	Description
0(0)	4	PARDCBAD	Address of DCB being opened/closed (same as PARDCBAB below)
0(0)	1	PAROPT	OPEN/CLOSE options
	1...	PARENLST	End of list
CLOSE Options			
	.1..	PARREWND	REWIND
	..11	PARLEAVE	LEAVE

Offset	Length	Name	Description
	..1.	PARFREE	Unallocate during CLOSE
	...1	PARRREAD	REREAD
OPEN Options			
	 1111	PAROUTPT	Output
	.... .111	PAROUTIN	Outin
	.... .1..	PARUPDAT	Update
	11	PARINOUT	Inout
	1	PARRDBCK	Readback
	....	PARINPUT	Input
1(1)	3	PARDCBAB	Address of DCB being open/closed

SAM OPEN/CLOSE Work Area—IGGSCW

This DSECT maps against the O/C/EOV work area fields DXCCW1-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(170)	8	DXCCW1	
368(170)	4	DXBLDL	BLDL parameter list -O
368(170)	4	SCWGETMA	Register save area for QSAM routines -C
368(170)	1	DXCCWOP	CCW OP code -O
368(170)	1	SCWSAVCD	Problem Determination Code -C
368(170)	1	DXUCSUCB	UCB UCS options -O
369(171)	3	DXCCWADR	Buffer address -O
369(171)	3	SCWGETMB	(Same as SCWGETMA) -C
372(174)	4	DXBLDLIM	Image name for BLDL -O
372(174)	2	DXCCWFLG	CCW flags -O
374(176)	2	DXCCWBYT	CCW byte count -O
376(178)	8	DXCCW2	
376(178)	4	DXIMGNAM	Image name -O
376(178)	4	SCWRALL	Save area for all registers -C
408(198)	8	DXCCW6	
408(198)	8	DXSAVUCS	Area to save UCS name -O
416(1A8)	8	DXCCW8	
416(1A8)	4	DXIMGDCB	Address of SYS1.IMAGELIB DCB -O
424(1B0)	8	DXCCW9	
424(1B0)	8	DXSAVFCB	Area to save FCB name -O
432(1B8)	8	DXCCW10	
432(1B8)	16	DXFCBUCS	UCS and FCB parameter fields -O
432(1B8)	8	DXFCBP	To clear FCB parameter field -O
432(1B8)	1	DXFCBSW1	Switch for FCB parameters -O
433(1B9)	1	DXABEND	Indicates DMABCOND to be issued -O
433(1B9)	1	DXFLAG1	FCB flag byte -O
434(1BA)	1	DXSTAGE2	Indicates next executor is a stage 2 executor -O
437(1BB)	1	DXFCBOPT	JFCB FCB options -O
438(1BC)	4	DXFCBID	FCB image identification -O
442(1C0)	8	DXCCW11	
442(1C0)	8	DXUCSP	Parameter list for UCS -O
442(1C0)	1	DXUCSSW1	Switch for UCS parameters -O
442(1C0)	1	DXABRETC	Internal return code for Problem Determination -O
443(1C1)	1	DXEROPT	To save DCBEROPT -O
444(1C2)	1	DXNABEND	Indicate ABEND in control -O
445(1C3)	1	DXUCSOPT	JFCB UCS options -O
446(1C4)	4	DXUCSID	UCS image identification -O
450(1C8)	8	DXCCW12	
450(1C8)	4	SCWXCTLP	Supervisor parameter list for XCTL -O/C

SAM/PAM/DAM GTRACE Buffer—IGGSPD

Offset	Length	Name	Description
0(0)	*	SPDBFR	SPD Buffer input record
0(0)	9	SPDHDR	Buffer header
0(0)	8	SPDDDNAM	DD name from TIOT
8(8)	1	SPDABCCD	ABEND condition code
9(9)	*	SPDTRACE	Trace record area (variable - maximum length is 247 bytes)
9(9)	*	SPDTRRCD	Trace record
9(9)	2	SPDRCDHD	Trace record header
9(9)	1	SPDRCDLN	Trace record length
10(A)	1	SPDBLKID	ID of trace record
11(B)	*	SPDDATA1	Block to be traced (no block address present)
11(B)	4	SPDBLKAD	Address of block to be traced (not present for block with ID less than 127)
15(F)	*	SPDDATA2	Block to be traced (block address present)

*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
Pointers and User Data Save Area			
0(0)	4	STWPARM	Address of user-supplied entry name (lower of two for change)
4(4)	4	STWHIGH	Address of higher of two user-supplied names (change only)
8(8)	2		Reserved
10(A)	2	STWOFFLW	Offset to add, replace, or delete location in low block
12(C)	8	STWOLDNM	Name of entry being deleted
20(14)	8	STWNEWNM	Name of new entry
28(1C)	3	STWTTR	Relative address of member
31(1F)	1	STWCTTRN	Alias bit, number of TTRNs, and length of user data
	1...	STWALIAS	This member name is an alias
32(20)	62	STWDATA	User data for entry
Flag, Condition, and Switch Bytes			
94(5E)	1	STWFLAG1	First flag byte bit definitions
	1		Reserved
	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
95(5F)	1	STWRTN	Return code save area
Control Blocks for STOW Channel Programs			
96(60)	4		If already on a doubleword go to the next fullword boundry
Event Control Block			
100(64)	4	ECBRB	RB (request block) address while waiting for event

Offset	Length		Name	Description
100(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 re-initiated.
		.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 ERP 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
101(65)	3		ECBRBA	Request block address (while awaiting completion of an event)
101(65)	3		ECBCCCNT	Zeroes or remainder of completion code (after completion of the event)
Prefix Sections of the IOB				
88(58)	8		IOBPREFX	Prefix sections
88(58)	8		IOBQSAMC	QSAM/BSAM/BPAM prefix
88(58)	8		IOBBSAMC	Chained scheduling
88(58)	8		IOBBPAMC	16 bytes
88(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(59)	1		IOBRSV05	Reserved
90(5A)	1		IOBCINOP	Offset of last I/O command for input operation (NOP CCW) from the ICB origin
91(5B)	1		IOBCONOP	Offset of last I/O command for output operation (NOP CCW) from the ICB origin
92(5C)	4		IOBCECB	Event control block
96(60)	4		IOBCICB	Address of first ICB on queue
100(64)	4		IOBCNOPA	Address of NOP command at end of queue
QSAM BSAM BPAM Prefix				
96(60)	8		IOBQSAMN	QSAM/BSAM/BPAM prefix
96(60)	8		IOBBSAMN	Normal scheduling
96(60)	8		IOBBPAMN	8 bytes
96(60)	4		IOBNIOBA	Address of next IOB on chain
96(60)	1		IOBNFLG1	Flag byte
		1...	IOBPRTOV	PRTOV occurred
		.1..	IOBWRITE	WRITE operation in process

Offset	Length		Name	Description
	..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(61)	3		IOBNIOBB	Address of the next IOB on the chain
100(64)	4		IOBNECB	Event control block address
Standard Section of the IOB				
104(68)	8		IOBSTDRD	
104(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 redundancy check needed (tape only)
	....	1...	IOBFCREX	FETCH command retry exit (direct access only)
	....	.1..	IOBIOERR	I/O error has occurred
	....	..1.	IOBUNREL	I/O request is unrelated (non-sequential)
	....	...1	IOBSPSVC	SAM/PAM flag set by SVC if I/O appendage should not process interrupt
105(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.	IOBSTATO	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(6A)	1		IOBSENS0	First sense byte
	1...	....	IOBS0B0	Bit 0 (device dependent)
	.1..	....	IOBS0B1	Bit 1 (device dependent)
	..1.	....	IOBS0B2	Bit 2 (device dependent)
	...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(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(6C)	4		IOBECBPT	Address of ECB to be posted upon completion of I/O

Offset	Length		Name	Description
108(6C)	1		IOBECBCC	Completion code for current I/O request
109(6D)	3		IOBECBPB	Address of ECB to be posted upon completion of I/O
112(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(71)	7		IOBCSW	Seven low-order bytes of CSW at channel end
113(71)	3		IOBCMDA	Command address (3890)
116(74)	2		IOBSTBYT	Status bits 32-47 (3890)
118(76)	2			Last two bytes of IOBCSW
120(78)	4		IOBSTART	Address of channel program
120(78)	1		IOBSIOCC	Bits 2 and 3 = condition code from SIO
121(79)	3		IOBSTRTB	Address of channel program
124(7C)	4		IOBDCBPT	Address of data control block for this IOB
124(7C)	1		IOBFLAG4	Flag byte
	1...	....	IOBGDPOL	Re-enter 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(7D)	3		IOBDCBPB	Address of data control block for this IOB
128(80)	4		IOBRESTR	Restart address for error retry
128(80)	1		IOBREPOS	Code used to reposition device
129(81)	3		IOBRSTRB	Restart address for error retry
132(84)	2		IOBINCAM	Value used to increment block count on tape
132(84)	1		IOBCRDCC	Optical reader—data check error count
133(85)	1		IOBCRILC	Optical reader—Incorrect Length error count
134(86)	2		IOBERRCT	Count of error retries
Direct Access Extension Section of the IOB				
136(88)	1		IOBM	Relative extent number for this request (0-15)
137(89)	2		IOBBD	Bin number (data cell)
137(89)	1		IOBBB1	
138(8A)	1		IOBBB2	
139(8B)	2		IOBCC	Cylinder number
139(8B)	1		IOBCC1	
140(8C)	1		IOBCC2	
141(8D)	2		IOBHH	Track number
141(8D)	1		IOBHH1	
142(8E)	1		IOBHH2	
143(8F)	1		IOBR	Record number
STOW Channel Programs				
144(90)	8		STWINCP	Channel program to read the initial two directory blocks
144(90)	8		STWSRCH1	Search ID equal
152(98)	8		STWTIC11	Transfer control to search ID
160(A0)	8		STWRDCT1	Read count
168(A8)	1		STWSRKY1	Search on key equal or high

169(A9)	3	STWKYAD1	Key address
172(AC)	4		Flags and byte count
176(B0)	8	STWTIC12	Transfer control to read count
184(B8)	8	STWRDAT1	Read data
192(C0)	8	STWRCKD1	Read count key data
200(C8)	8	STWWRDCP	Channel program to write and read directory blocks
200(C8)	1	STWSRCH2	Search ID equal
201(C9)	3	STWIDAD2	ID address
204(CC)	4		Flags and byte count
208(D0)	8	STWTIC2	Transfer control to search ID
216(D8)	1	STWWRKD1	Write key and data
217(D9)	3	STWWRAD2	Write address
220(DC)	1	STWWFLG2	Flags
	.1..	STWCMDCH	Command chain to next CCW
221(DD)	3		Byte count
224(E0)	1	STWRCKD2	Read count key data
225(E1)	3	STWRDAD2	Read Address
228(E4)	4		Flags and byte count
232(E8)	1	STWRCKD3	Read count key data
233(E9)	3	STWRDAD3	Read address
236(EC)	4	STWRDFL3	Flags and byte count
240(F0)	4	STWRCKD4	Read count, key, data
244(F4)	4	STWRDFL4	Flags and byte count

STOW Input/Output Buffers
(For details see Buffer DSECT below)

248(F8)	276	STWBUF1	Initially contains the first of two directory blocks read
524(20C)	276	STWBUF2	Initially contains the second of two directory blocks read
800(320)	276	STWBUF3	Initially used as the first output buffer
1076(434)	276	STWBUF4	Temporarily used as first input buffer for EXCP.
1352(548)	8	STWEND	End of work area

Map of STOW Input/Output Buffers

0(0)	8	BUFCNT	Count field containing absolute disk address
0(0)	5	BUFCCHHR	CCHHR field
5(5)	3	BUFKDD	Key and data length
8(8)	8	BUFKEY	Key field (highest member name)
16(10)	256	BUFDATA	Data Area
16(10)	2	BUFN	Number of bytes used in this directory block
18(12)	254	BUFENTRY	Directory entries
272(110)	4	BUFADDR	Used to chain buffers

SVRB Extended Save Area

0(0)	4	XSAREG4	Save area for register 14
4(4)	4	XSASTWWA	Address of STOW work area
8(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(0)	72	SYNSAVE	Save area
0(0)	4	SYNPL1	Used by PL/1 language program
4(4)	4	SYNPREV	Address of previous save area
8(8)	4	SYNNEXT	Address of next save area
12(C)	60	SYNGRS	General register save area
72(48)	8	SYNVLFLD	Length field for variable-length records
80(50)	1	SYNMSG	Data Area

Offset	Length	Name	Description
80(50)	4	SYNRDERR	Return information if read error
84(54)	2	SYNBYTRD	Number of bytes read
86(56)	35	SYNWAREA	Work area
86(56)	1	SYNPURG	Error type indicator
86(56)	1	SYNACMTH	Access method input code
87(57)	1		Unused
88(58)	8	SYNWRKA	Work Area
88(58)	4	SYNWKA1	Work area number 1
92(5C)	4	SYNWKA2	Work area number 2
96(60)	4		Unused
100(64)	20	SYNWORK	Work area
120(78)	1	SYNSTART	Blank
121(79)	1	SYNCMMA1	Comma
122(7A)	8	SYNJOBNM	Job name
130(82)	1	SYNCMMA2	Comma
131(83)	8	SYNSTPNM	Step name
139(8B)	1	SYNCMMA3	Comma
140(8C)	3	SYNUNTID	Unit address
143(8F)	1	SYNCMMA4	Comma
144(90)	2	SYNDVTYP	Device type
146(92)	1	SYNCMMA5	Comma
147(93)	8	SYNDDNM	DD name
155(9B)	1	SYNCMMA6	Comma
156(9C)	6	SYNOPRTN	Operation attempted
162(A2)	1	SYNCMMA7	Comma
163(A3)	15	SYNERROR	Error description
178(B2)	1	SYNCMMA8	Comma
179(B3)	14	SYNPOS	Area to unpack ICB seek address
179(B3)	7	SYNPOSM1	Unused—Magnetic tape
186(BA)	7	SYNPOSM2	Area to unpack block count for magnetic tape
186(BA)	6	SYNPOSMV	Unpack value
192(C0)	1	SYNPOSMS	Sign byte in unpack format
193(C1)	1	SYNCMMA9	Comma
193(C2)	5	SYNACCSS	Access method type
199(C7)	1	SYNBLNK2	Blank
200(C8)	4	SYNPRMR1	Parameter register save area
204(CC)	4	SYNPRMR2	Parameter register save area
208(D0)	4	SYNEND	End of IGGSYN

SYNADAF and SYNADRLS SVRB Extended Save Area

160(A0)	4	SYNRETA	Return address
164(A4)	16	SYNXCTPL	XCTL parameter list
164(A4)	4	SYNXCTEP	Address of the entry point name
168(A8)	12	SYNXCTLT	List of parameters
168(A8)	4	SYNXCTDB	Address of the DCB
172(AC)	8	SYNXCTNM	Entry point name
176(B0)	1	SYNXCTID	Load module ID
180(B4)	1	SYNESTPL	Flags for TCB PURGE and ASYNCH
181(B5)	3		Exit address not specified
184(B8)	4		Parameter list address not specified
188(BC)	4		TCB not specified
192(C0)	1		Flags
193(C1)	3		Reserved
196(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
198(C6)	1	SYNURKEY	User key
199(C7)	9	SYNUNPKA	Work area for unpack

SETPRT Parameter List—IHASPP

Offset	Length	Name	Description
0(0)	1		Unused
1(1)	3	SPPDCBB	DCB address
4(4)	4	SPPUCS	UCS module ID (not used for 3800)
8(8)	1	SPPLDMOD	Flag byte describing UCS load mode (not used for 3800)
	.0..		Load mode=no fold
	.1..		Load mode=fold
	x.xx xxxx		Unused
9(9)	1	SPPVERFY	Flag byte describing UCS verify (not used for 3800)
	...0		No verification for UCS
	...1		UCS verification requested
	xxx. xxxx		Unused
10(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	Unfold 3203 or 3211 UCS
	.... 01..	SPPFOLD	Fold 3203 or 3211 UCS
	.... ..x.		Unused
	1	SPPEXTL	SETPRT parameter list is at least 48 bytes long
11(B)	4	SPPFCB	FCB module ID or address of in-storage FCB module (see SPPFLAG2)
15(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)
	1		Issue WTOR for forms alignment (not used for 3800)

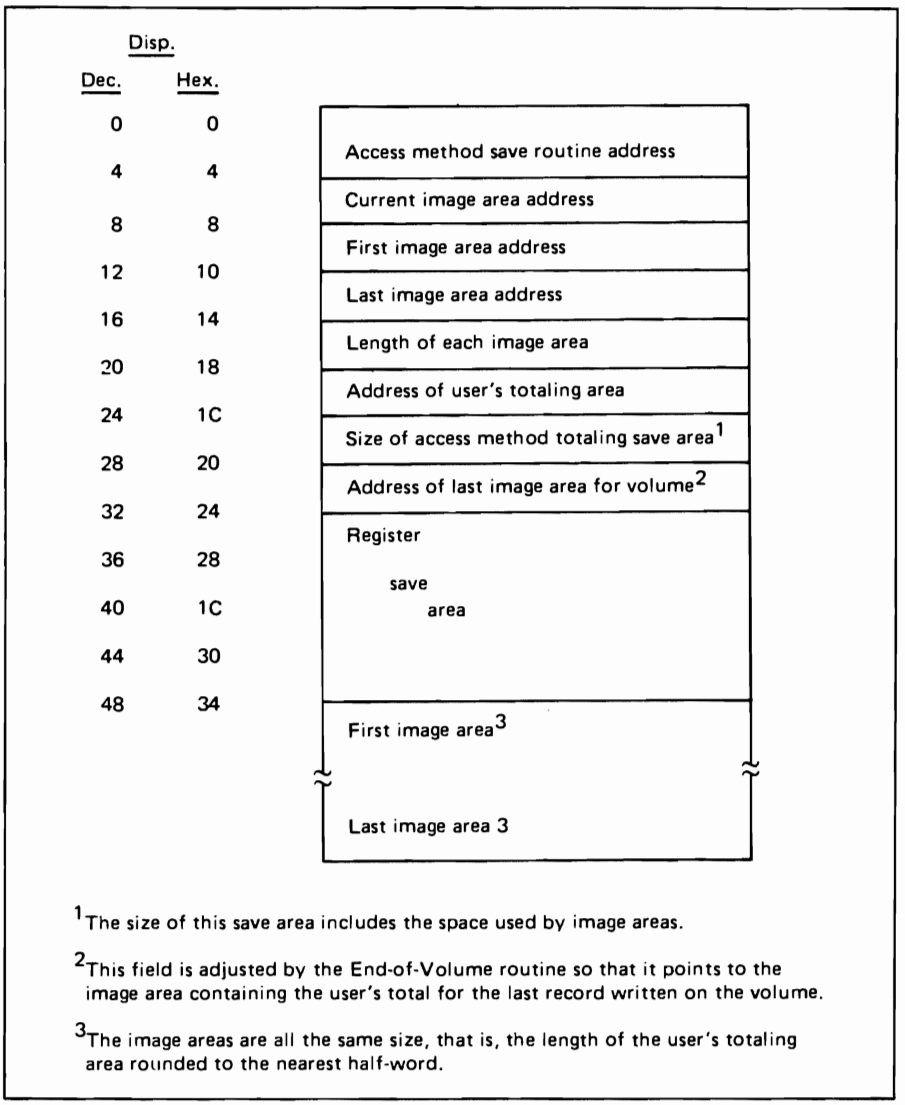
From this offset on, the fields apply only to the IBM 3800 Printing Subsystem.

16(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
	.... .1..	SPPBTREQ	Bypass threading change WTOR
	.... ..1.	SPPBOMSG	Bypass WCGM's exceeded error message
	1	SPPFORC	JES force load of the FCB

Offset	Length	Name	Description
17(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 if specified is a module ID.
	.... .1..		FCB is specified as an address.
	xx		Unused bits.
18(12)	1	SPPCPYNR	The number of copies to be printed.
19(13)	1	SPPSTCNR	The copy number of the first copy to be printed.
20(14)	2	SPPLEN	Length of the parameter list.
22(16)	1	SPPFRMNR	The number of copies to be forms flashed, starting with the first printed copy.
23(17)	1	SPPTRC	The character arrangement table to be selected when the specified copy modification record is loaded in the printer.
24(18)	4	SPPMODPT	The module ID or in-storage address of the copy modification module.
28(1C)	4	SPPIMAGE	The identifier for the forms overlay frame.
32(20)	4	SPPXLAT1	The module ID or in-storage address of character arrangement table module 0.
36(24)	4	SPPXLAT2	The module ID or in-storage address of character arrangement table module 1.
40(28)	4	SPPXLAT3	The module ID or in-storage address of character arrangement table module 2.
44(2C)	4	SPPXLAT4	The module ID or in-storage address of character arrangement table module 3.
48(30)	4	SPPEMSGA	Address of the message communication area for error information.
52(34)	4	SPPLIDCB	Address of the library DCB for 3800 load modules.

Access Method Save Area for User Totaling

The access method save area for user totaling is pointed to by the address in bytes 5-7 in the EXCP access method, BSAM, or QSAM-dependent section of the DEB.



[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.

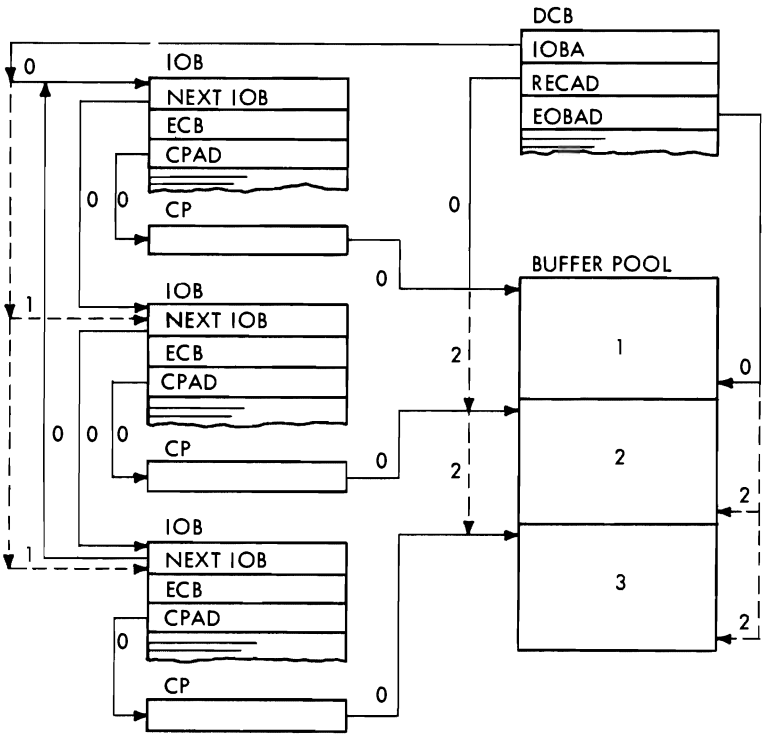
Figure 35. Access Method Save Area for User Totaling

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 *OS/VS2 Open/Close/EOV Logic*. Note that the access method executors do not have transfer control tables like common Open and Close modules do.

QSAM Control Blocks

Figure 36 shows the control blocks used in QSAM. Through the data control block (DCB), the QSAM routines associate the data being processed with the processing program. Fields in the DCB point to the start of a buffer, the end of a buffer, and an input/output block (IOB). These fields are updated as successive channel programs are executed. Each IOB points at the next IOB and at a channel program (CP), and carries an event control block (ECB) that the I/O supervisor posts after the channel program has been executed.



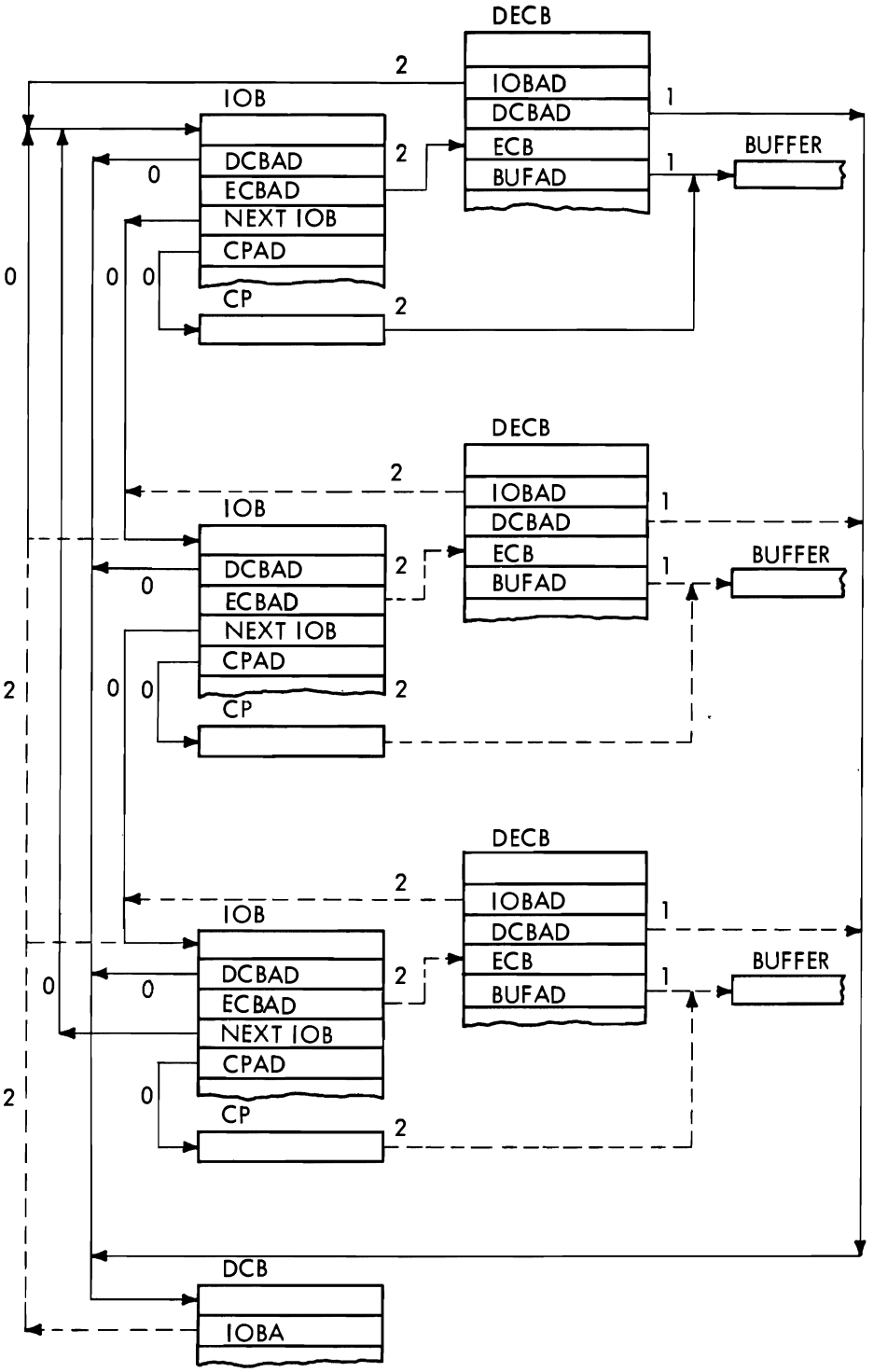
Legend:
Address Values:
0 Entered by the OPEN executor.
1 Updated by the synchronizing routine.
2 Updated by the GET or PUT routine.
– – – Successive Address Values

Figure 36. QSAM Control Blocks

BSAM Control Blocks

Figure 37 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 Provided by the processing program.

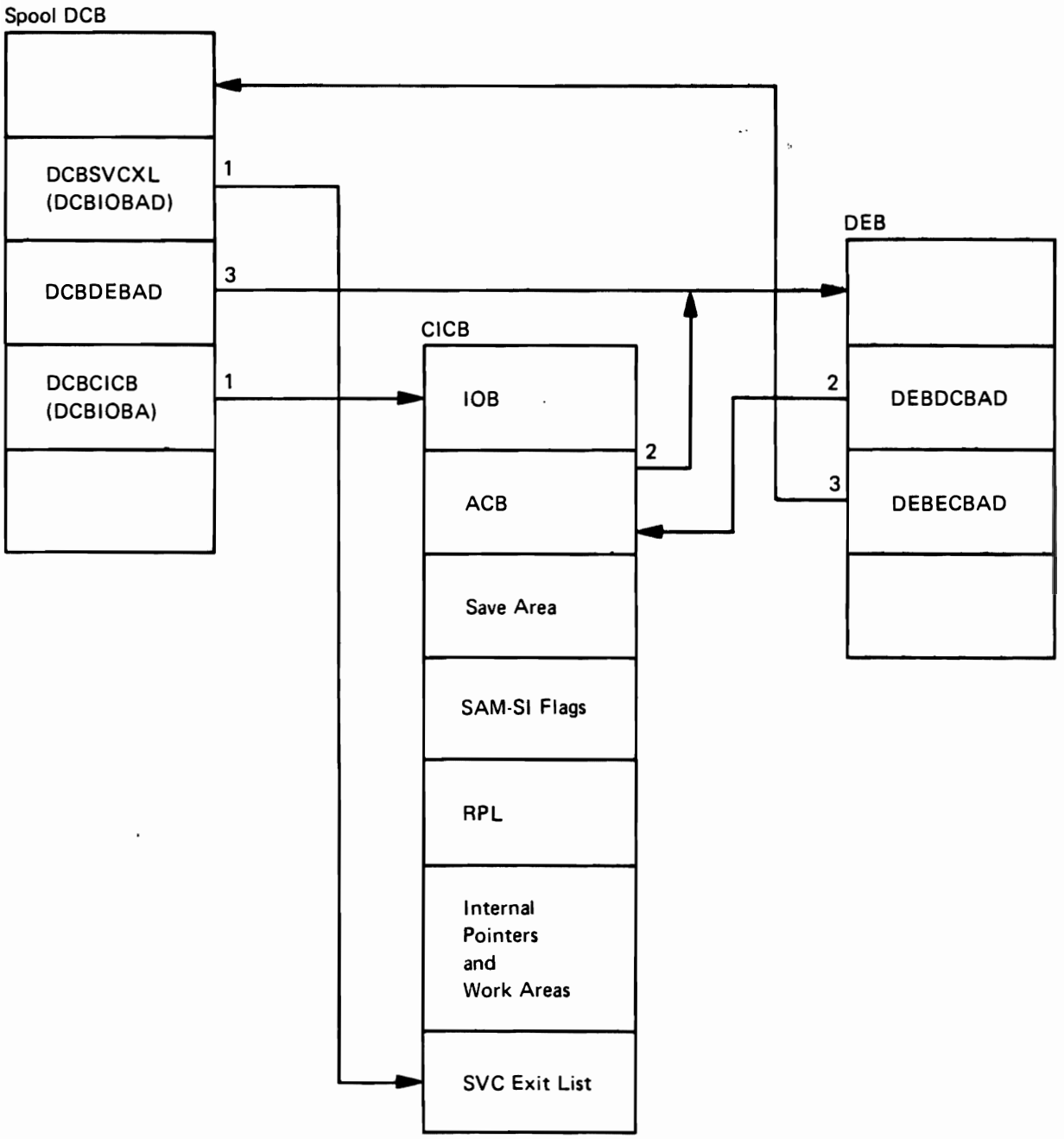
2 Completed by the READ or WRITE routine.

--- Successive Address Values.

Figure 37. BSAM Control Blocks

JES Compatibility Interface Control Block (CICB)

See Figure 38 for the relationship of the SYSIN/SYSOUT data sets.



Notes:

1. Set by DCB open executors (SAM-SI).
2. Set by ACB open (JES2 open executors).
3. Set by DCB open executors after ACB open.

Figure 38. 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 *OS/VS Message Library: VS2 System Messages* and *OS/VS Message Library: VS2 System Codes*.

ABEND Code	Return Code	Module	Message Number
001		IGG019AH	
		IGG019AN	
002	04	IGG019AB	IEC036I
		IGG019AD	
		IGG019AE	
		IGG019BN	
		IGG019BO	
		IGG019FB	
		IGG019FD	
		IGG019FF	
	08	IGG019CD	
		IGG019CV	
		IGG019CC	
		IGG019TD	
		IGG019TV	
		IGG019T2	
	0C	IGG019CD	
		IGG019CV	
		IGG019TD	
		IGG019TV	
	10	IGG019FG	
		IGG019C2	
	14	IGG019AL	
		IGG019DK	
		IGG019C2	
		IGG019T2	
	18	IGG019AJ	
		IGG019BP	
		IGG019DJ	
		IGG019FJ	
003	01	IGG019CC	
	02	IGG019CE	
		IGG019CF	
		IGG019FK	
	03	IGG019FA	
		IGG019FQ	
004	01	IGG0197N	
	02	IGG0197N	
	03	IGG0197N	
	04	IGG0197N	
	05	IGG0197M	
	06	IGG0197Q	
008		IGG019BS	
		IGG019DC	

ABEND Code	Return Code	Module	Message Number
013	10	IGG0191C	IEC141I
	14	IGG0191B	
	18	IGG0191B	
	1C	IGG0191B	
	20	IGG0191A	
	24	IGG0191A	
		IGG0199G	
	28	IGG0191A	
		IGG0199G	
	30	IGG0191F	
	34	IGG0191I	
		IGG0199G	
	38	IGG0191H	
		IGG0199K	
	3C	IGG0191D	
	40	IGG01910	
		IGG01990	
	44	IGG0191K	
		IGG0191W	
	48	IGG01911	
	4C	IGG0191A	
		IGG0196B	
		IGG0199G	
	50	IGG0196B	
	54	IGG0196A	
	58	IGG01912	
		IGG0191I	
	5C	IGG01915	
		IGG01916	
	60	IGG0191A	
	70	IGG0199G	
	88	IGG0191B	
	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	
	D0	IGG0191A	
112	01	IGCT0018	IEC908I
	02	IGCT0018	
	03	IGCT0018	
	04	IGCT0018	
	13	IGCT0018	
	14	IGCT0018	
115		IGG0210A	
118	01	IGCT002D	IEC912I
	02	IGCT002D	

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	
212	01	IGCT0018	IEC909I
	02	IGCT0018	
	03	IGCT0018	
215	01	IGCT0021	IEC910I
	02	IGCT0021	
	03	IGCT0021	
	04	IGCT0021	
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
413	2C	IGG0191N	IEC145I
444		IGCT006H	
451	01	IGCT1081	IEC918I
	02	IGCT1081	
	03	IGCT1081	
	04	IGCT1081	
544		IGCT006H	
644		IGCT006H	

ABEND Code	Return Code	Module	Message Number
744		IGCT006H	IEC907I
B13	04	IGG0191U	IEC152I
		IGG0196R	
	08	IGG0191U	
	0C	IGG0191V	
		IGG0196R	
	10	IGG0197U	
	14	IGG0197U	
	18	IGG0197F	
	1C	IGG0197F	
	20	IGG0191T	
		IGG0196R	
	24	IGG0191T	
		IGG0196R	
	28	IGG0191T	
	2C	IGG0197E	
	30	IGG0197E	
	34	IGG0197E	
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 Codes and Messages (For 3800 Only)

The return codes produced by the SETPRT executors that support the IBM 3800 Printing Subsystem 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 the "Data Areas" section for a description of the SETPRT work area).

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 IGG08111 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).
0000030		IEC176I	There are more character set IDs requested in the character arrangement tables than the number of WCGMs installed on the printer.
0000044		IEC182I	A position in the translate table refers to a WCGM for which no character set has been specified.
000004C	xx000018	IEC184I	A load check occurred while loading a library character set (xx indicates the opcode of the CCW when the error occurred).
000004C	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 modules within that table).
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 modules 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 modules within that table, and xx indicates the 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).
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 a 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 cancelled 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 cancelled SETPRT processing after receiving a request to load a Forms Overlay negative.
0000002C		IEC172I	The operator cancelled 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 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-required 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 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.

SAM Register Saving Convention

The user is responsible for providing SAM with an 18-word Save Area. This address of the Save Area must be passed in register 13. The Read/Write GET/PUT modules will save the user's registers 14 through 8 at the Save Area plus 20 (X '14'). In a formatted dump, registers 14 through 8 will be stored at register locations labeled R1 through R10 respectively.

APPENDIX A: PAPER TAPE CODE CONVERSION ROUTINES

Get routine IGG019AT (paper tape) and Write routine IGG019BF (paper tape) use the tables in the following modules to convert characters read from paper tape to EBCDIC characters.

Code Conversion Module IGG019CM

This module is loaded by the Open executor if the DCB specifies paper tape, and code conversion for Teletype* transmission code.

The module consists of three tables:

- A validity-checking and special functions table
- A lowercase character translation table
- An uppercase character translation table

Code Conversion Module IGG019CN

This module is loaded by the Open executor if the DCB specifies paper tape, and code conversion for ASCII paper tape code.

The module consists of two tables:

- A validity-checking and special functions table
- A character translation table

Code Conversion Module IGG019CO

This module is loaded by the Open executor if the DCB specifies paper tape and code conversion for Burroughs paper tape code.

The module consists of two tables:

- A validity-checking and special functions table
- A character translation table

Code Conversion Module IGG019CP

This module is loaded by the Open executor if the DCB specifies paper tape and type and code conversion for Friden paper tape code.

The module consists of three tables:

- A validity-checking and special functions table
- A lowercase character translation table
- An uppercase character translation table

*Trademark of Teletype Corporation

Code Conversion Module IGG019CQ

This module is loaded by the Open executor if the DCB specifies paper tape and code conversion for IBM PTTC/8 code.

The module consists of three tables:

- A validity-checking and special functions table
- A lowercase character translation table
- An uppercase character translation table

Code Conversion Module IGG019CR

This module is loaded by the Open executor if the DCB specifies paper tape and code conversion for NCR paper tape code.

The module consists of three tables:

- A validity-checking and special functions table
- A lowercase character translation table
- An uppercase character translation table

Translate Routine IGC0010C

This routine is a type 2 SVC routine if made resident at system generation or a type 3 SVC routine if it is transient that translates data from EBCDIC to ASCII on output and ASCII to EBCDIC on input. It is entered when an XLATE macro instruction (SVC 103) is issued.

The routine must be given the following parameters:

- Register 0 = Length of area to be translated
- Register 1

  Byte 0—bit 0 = 0: Translate from ASCII to EBCDIC

  bit 0 = 1: Translate from EBCDIC to ASCII

  bits 1-7 : Not used

  Byte 1-3: Address of area that contains record to be translated.

APPENDIX B: BSAM/QSAM CHANNEL PROGRAMS

CHANNEL PROGRAM FOR OUTPUT, SIMPLE BUFFERING (IGG0191D)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	WRT CKD	CCW8 [1]	DS	8
CCW4		DCBBUFCB [2]	C[4]S	DCBBLKSI [2]
	(WRITE CHECK)			
CCW5	SCH ID EQ	CCW8	C	5
CCW6	TIC	*-8		
CCW7	READ KD		K	
CCW8	CCHHRKDD [1]			
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	WRT CKD	CCW11 [3]	DS	8
CCW5		DCB BUFCB [2]	DS	DCBBLKSI [2]
CCW6	RD SECTOR	SECTOR2	C[4]	1
	(WRITE CHECK)			
CCW7	SET SECTOR	SECTOR2	C	1
CCW8	SCH ID EQ	CCW11	C	5
CCW9	TIC	*-8		
CCW10	RD KD		K	
CCW11	CCHHRKDD [3]			

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip

1. If there is no write check, CCHHRKDD will be in CCW5.
2. For QSAM only.
3. If there is no write check, CCHHRKDD will be in CCW7.
4. For write-check only.

CHANNEL PROGRAM FOR TRACK-OVERFLOW, INPUT, NON-FORMAT-U (IGG0191H)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	RD DATA		CSK	7
CCW4	RD CNT	DCBFDAD+3[1]	CS	5
CCW5	RD DATA[2]	BUFAD[3]	S[4]	
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	RD DATA		CSK	7
CCW5	RD CNT	DCBFDAD+3[1]	C	5
CCW6	RD DATA	BUFAD[3]	CS	
CCW7	RD SECTOR	SECTOR2		1

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip

1. If there is more than one IOB, then address is IOBSEEK+3 of next IOB.
2. If keys are specified, READ KD.
3. For QSAM only.
4. SLI bit is on for format-V records.

Note: Format-U with track-overflow is not supported by RPS.

CHANNEL PROGRAM FOR OUTPUT, TRACK-OVERFLOW, NON-FORMAT-U (IGG0191H)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8	TIC	ADDR CCW7
CCW3	WRT S CKD[1]	A COUNT AT END OF C.P.	D	8
CCW4	null	DATA ADDR	C[2]	KEY+SEGMENT DATA LNTH[3]
CCW5[4]	M/T SCH ID EQ[5]	SRCH ADDR AT END OF C.P.	C	5
CCW6	TIC	*-8	0	SRCH ADDR IN CCW5
CCW3				
CCW4	(WRITE CHECK)			
CCW7	SEEK HEAD/NOP[6]	IOBSEEK+1	C	6
CCW8	SCH ID EQ	ADDR OF FIRST SEG COUNT FLD	C	5
CCW9	TIC	*-8		
CCW10	RD KD	0	K	

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip

1. Op code is WRT CKD for the last record segment.
2. Chaining is turned off in this CCW of the last segment if no validity checking is specified.
3. The key is written on the first record segment only.
4. CCWs 5 and 6 are required if more than one segment is to be written. Each segment but the first is written by CCWs 5, 6, 3, 4, in that order.
5. CCW5 op code and address may be overlaid with the parasitic TIC from CCW2 if not all the channel program segments are needed, but validity-checking is specified.

CHANNEL PROGRAM FOR TRACK-OVERFLOW, OUTPUT, NON-FORMAT-U (IGG0191H)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8	TIC	ADDR CCW8
CCW4	WRT S CKD[1]	COUNT AT END OF C. P.	D	8
CCW5	null	DATA ADDR	C	KEY+SEG DATA LNTH[3]
CCW6[4]	M/T SCH ID EQ[5]	SRCH ADDR AT END OF C. P.	C	5
CCW7	TIC	*-8	0	ADDR OF SRCH ARG IN CCW5
CCW4				
CCW5				
CCW8	RD SECTOR (WRITE CHECK)	SECTOR2	C[2]	1
CCW9	SEEK CYL	IOBSEEK+1	C	6
CCW10	SET SECTOR	SECTOR2	C	1
CCW11	SCH ID EQ	ADDR FRST SEG CNT	C	5
CCW12	TIC	*-8		
CCW13	RD KD	0	K	

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip

1. The op code is WRT CKD for the last record segment.
2. Chaining is turned off in this CCW of the last segment if no validity checking is specified.
3. The key is written on the first record segment only.
4. CCWs 6 and 7 are required if more than one segment is to be written. Each segment but the first is written by CCWs 6, 7, 4, and 5. The last segment is followed by the parasitic TIC to CCW8 if not all the channel program segments are used.
5. CCW6 op code and address may be overlaid with the parasitic TIC from CCW3 if not all the channel program segments are needed.

CHANNEL PROGRAM FOR INOUT, OUTIN (IGG0191J and IGG0196L)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	M/T RD DATA		CSK	1
CCW4	M/T RD CNT	NEXT IOBSEEK+3[1]	CS	5
CCW5	M/T RD KD[2]		S[3]	
CCW6	SCH ID EQ	IOBSEEK+3	C	5
CCW7	TIC	*-8		
CCW8	WRT CKD	CCW13	DS	8
CCW9	00 (WRITE CHECK)		S	
CCW10	SCH ID EQ	CCW13	C	5
CCW11	TIC	*-8		
CCW12	M/T RD KD		SK	
CCW13	additional search argument			

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING AND WITH SEARCH-DIRECT				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	TIC	*+8		
CCW4	M/T RD KD[2]		CS	
CCW5	M/T RD CNT		C	
CCW6	00			
CCW7	00			
CCW8	SCH ID EQ		C	
CCW9	TIC	*-8		
CCW10	WRT CKD			
CCW11	00			
CCW12	00			
CCW13	SCH ID EQ	CCW13	C	5
CCW14	TIC	*-8		
CCW15	M/T RD KD		SK	

FLAGS

D = Data Chain
C = Common Chain
S = SLI
K = Skip

1. If there is only one IOB, the address is the DCBFDAD+3.
2. Read both key and data only if key is specified.
3. SLI bits is on for format-U and format-V records.

CHANNEL PROGRAM FOR INOUT, OUTIN (IGG0191J and IGG0196L)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	M/T RD DATA		CSK	1
CCW5	M/T RD CNT	CCW19	C	8
CCW6	M/T RD KD[1]		CS	
CCW7	RD SECTOR	SECTOR2		1
CCW8	SET SECTOR	SECTOR1	C	1
CCW9	SCH ID EQ	IOBSEEK+3	C	5
CCW10	TIC	*-8		
CCW11	WRT CKD	CCW18	DS	8
CCW12	00		CS	
CCW13	RD SECTOR (WRITE CHECK)	SECTOR2	C	1
CCW14	SET SECTOR	SECTOR2	C	1
CCW15	SCH ID EQ	CCW18	C	5
CCW16	TIC	*-8		
CCW17	M/T RD KD		SK	
CCW18	additional search argument			
CCW19	additional search argument			

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITH ROTATIONAL POSITION SENSING AND WITH SEARCH-DIRECT				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	TIC	* +8		
CCW5	M/T RD KD[1]		CS	
CCW6	M/T RD CNT	CCW19	C	8
CCW7	RD SECTOR	SECTOR2		1
CCW8	SET SECTOR	SECTOR1	C	1
CCW9	SCH ID EQ	IOBSEEK+3	C	5
CCW10	TIC	*-8		
CCW11	WRT CKD	CCW18	DS	8
CCW12	00		CS	
CCW13	RD SECTOR (WRITE CHECK)	SECTOR2	C	1
CCW14	SET SECTOR	SECTOR2	C	1
CCW15	SCH ID EQ	CCW18	C	5
CCW16	TIC	*-8		
CCW17	M/T RD KD		SK	
CCW18	additional search argument			
CCW19	additional search argument			

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip

1. Read both key and data only if key is specified.

CHANNEL PROGRAM FOR CHAINED SCHEDULING DIRECT-ACCESS OUTPUT WITH WRITE-CHECK (IGG0191K)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	M/T SCH ID EQ	ICBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	WRT CKD	CCW9	DSP	8
CCW4	00	BUFFER	CS	BUF LENGTH
CCW5	SCH ID EQ	CCW9	C	5
CCW6	TIC	*-8		
CCW7	M/T RD DATA[1]		CK	1
CCW8	NOP	CCW1 or 3 in NEXT ICB	S	1
CCW9	CCHHRKDD			
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR = 0	C	1
CCW2	READ HA		CK	5
CCW3	M/T SCH ID EQ	ICBSEEK+3	C	5
CCW4	TIC	*-8		
CCW5	WRT CKD	CCW13	DSP	8
CCW6	00	BUFFER	CS	BUF LENGTH
CCW7	READ SECTOR	SECTOR 2	C	1
CCW8	SET SECTOR	SECTOR 2	C	1
CCW9	SCH ID EQ	CCW13	C	5
CCW10	TIC	*-8		
CCW11	M/T READ DATA[1]		CK	1
CCW12	NOP	CCW 1 or 5 in NEXT ICB	S	1
CCW13	CCHHRKDD			

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip
P = PCI

1. If keys are specified, READ KD.

CHANNEL PROGRAM FOR STANDARD FORMAT-F INPUT, SIMPLE BUFFERING (IGG0191O)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	READ KD [1]	DCBBUFCB [2]		DCBBLKSI [2]
WITH ROTATIONAL POSITION SENSING				
CCW1	SET S	SECTOR 1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	READ KD [1]	DCBBUFCB [2]	C	DCBBLKSI [2]
CCW5	M/T READ CNT		CSK	1
CCW6	READ S	SECTOR 2		1

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip

1. If key is not specified, read data only.
2. For QSAM only.

CHANNEL PROGRAM FOR NOT-STANDARD FORMAT-F SIMPLE BUFFERING, INPUT (IGG0191O)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	TIC	*+8	CSK	
CCW4	RD COUNT	NEXT IOBSEEK+3[2]	CS	5
CCW5	RD KD[3]	DCB BUFCB[1]	S	DCBBLKSI[1]
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	TIC	*+8	CSK	
CCW5	RD COUNT	CCW8	C	8
CCW6	RD KD[3]	DCBBUFCB[1]	CS	DCBBLKSI[1]
CCW7	RD SECTOR	SECTOR2		
CCW8	CCHHRKDD			

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip

1. For QSAM only.
2. If there is one IOB, then address is DCBFDAD+3.
3. If the data set has no keys, then the op code is RD D.

CHANNEL PROGRAM FOR NOT-STANDARD FORMAT-F, SIMPLE BUFFERING, INPUT, SEARCH-DIRECT (IGG0199O)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQU	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	TIC[1]	*+8[1]	C	8
CCW4	M/T RD D[2,3]		C	D_L
CCW5	M/T CNT	*+8		
CCW6	CCHHR$K_L D_L D_L$			
CCW7	CCHHR$K_L D_L D_L$			
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	CS	1
CCW2	SCH ID EQU	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	TIC[1]	*+8[1]	C	
CCW5	M/T RD D[2,3]	BUFFER	CS	D_L
CCW6	M/T RD CNT	*+16	C	8
CCW7	READ SECTOR	SECTOR2		1
CCW8	CCHHR$K_L D_L D_L$			
CCW9	CCHHR$K_L D_L D_L$[1]			

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip
P = PCI

1. For a search on record 0, this CCW is changed to an M/T RD CNT and its address field is CCW7.
2. If keys are present, the op code is RD KD and count is $K_L + D_L$.
3. For exchange buffering, there will be as many RD D CCWs as the blocking factor.

APPENDIX C: UPDATE CHANNEL PROGRAMS

CHANNEL PROGRAM FOR DIRECT-ACCESS, UPDATE, NO TRACK-OVERFLOW (IGG0196P)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	WRT DATA[1] (WRITE CHECK)[8]	BUFFER[2]	CS[3]	DCBBLKSI[2]
CCW4	SCH ID EQ	IOBSEEK+3	C	5
CCW5	TIC	*-8		
CCW6	RD DATA[4]		C[5]S[3]K	DCBBLKSI[2]
CCW7	SEEK HEAD/NOP[6]	CCW13+1	C	6
CCW8	SCH ID EQ	CCW13+3	C	5
CCW9	TIC	*-8		
CCW10	TIC	*+8		
CCW11	RD COUNT	NEXT CCW13[7]	CS	5
CCW12	RD DATA[4]	BUFFER[2]	S[3]	DCBBLKSI[2]
CCW13	MBBCCHHR			

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip

1. If key is specified, WRT KD.
2. For QSAM only.
3. SLI bit is on for format-U and format-V records.
4. If key is specified, READ KD.
5. Command chain off for BSAM.
6. CCW7 is present for QSAM only. Op code is SEEK HEAD for cylinder allocation and NOP for track allocation.
7. If there is only one IOB, the address field is DCBFDAD+3.
8. CCWs 4-6 are present for write-check.

CHANNEL PROGRAM FOR DIRECT-ACCESS UPDATE, NO TRACK-OVERFLOW (IGG0191Z, IGG01923)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR A	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	WRT DATA[1] (WRITE CHECK)[8]	BUFFER	C[7]S	DCBBLKSI[2]
CCW5	SET SECTOR	SECTOR A	C	1
CCW6	SCH ID EQ	IOBSEEK+3	C	5
CCW7	TIC	*-8		
CCW8	RD DATA[3]		C[4]SK	DCBBLKSI[2]
CCW9	SEEK HEAD/NOP[5]	CCW17+1	C	6
CCW10	SET SECTOR	SECTOR1	C	1
CCW11	SCH ID EQ	CCW17+3	C	5
CCW12	TIC	*-8		
CCW13	TIC	*+8		
CCW14	RD COUNT	CCW18[6]	C	8
CCW15	RD DATA	BUFFER[2]	CS	DCBBLKSI[2]
CCW16	RD SECTOR	SECTOR2		1
CCW17	MBBCCHHR			
CCW18	CCHHRKDD			

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip

1. If key is specified, WRT KD.
2. For QSAM only.
3. If key is specified, READ KD.
4. Command chain is off for BSAM.
5. CCW9 is present for QSAM only. Op code is SEEK HEAD for cylinder allocation and NOP for track allocation.
6. If there is only one IOB, the address field is DCBFDAD+3.
7. Command chain is off for BSAM if write-check is not specified.
8. CCWs 5-8 are present for write-check.

CHANNEL PROGRAM FOR DIRECT-ACCESS UPDATE, TRACK-OVERFLOW (IGG0196P)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	WRT DATA[1] (WRITE CHECK)[8]	BUFFER[2]	CS[3]	DCBBLKSI[2]
CCW4	SEEK HEAD/NOP[9]	IOBSEEK+1	C	6
CCW5	SCH ID EQ	IOBSEEK+3	C	5
CCW6	TIC	*-8		
CCW7	RD DATA[4]		C[5]S[3]K	DCBBLKSI[2]
CCW8	SEEK HEAD/NOP[6,9]	CCW14+1	C	6
CCW9	SCH ID EQ	CCW14+3	C	5
CCW10	TIC	*-8		
CCW11	RD DATA		CSK	50 (non-zero length)
CCW12	RD COUNT	NEXT CCW14[7]	CS	5
CCW13	RD DATA[4]	BUFFER[2]	S[3]	DCBBLKSI[2]
CCW14	MBBCCHHR			

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip

1. If key is specified, WRT KD.
2. For QSAM only.
3. SLI bit is on for format-U and format-V records.
4. If key is specified, READ KD.
5. Command chain is off for BSAM.
6. CCW8 is present for QSAM only.
7. If there is only one IOB, the address field is DCBFDAD+3.
8. CCWs 4-7 are present for write-check.
9. Op code is SEEK HEAD for cylinder allocation and NOP for track allocation.

CHANNEL PROGRAM FOR DIRECT-ACCESS UPDATE, TRACK-OVERFLOW (IGG0191Z, IGG01923)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR A	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	WRT DATA[1] (WRITE CHECK)[7]	BUFFER	C[6]S	DCBBLKSI[2]
CCW5	SEEK HEAD/NOP[8]	IOBSEEK+1	C	6
CCW6	SET SECTOR	SECTOR A	C	1
CCW7	SCH ID EQ	IOBSEEK+3	C	5
CCW8	TIC	*-8		
CCW9	RD DATA[3]		C[4]SK	DCBBLKSI[2]
CCW10	SEEK HEAD/NOP[5,8]	CCW18+1	C	6
CCW11	SET SECTOR	SECTOR1	C	1
CCW12	SCH ID EQ	CCW18+3	C	5
CCW13	TIC	*-8		
CCW14	RD DATA		CSK	50 (non-zero length)
CCW15	RD COUNT	NEXT CCW18	CS	5
CCW16	RD DATA	BUFFER[2]	CS	DCBBLKSI[2]
CCW17	RD SECTOR	SECTOR2		1
CCW18	MBBCCHHR			

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip

1. If key is specified, WRT KD.
2. For QSAM only.
3. If key is specified, READ KD.
4. Command chain is off for BSAM.
5. CCW10 is present for QSAM only.
6. Command chain is off for BSAM if write-check is not specified.
7. CCWs 5-9 are present for write-check.
8. Op code is SEEK HEAD for cylinder allocation and NOP for track allocation.

CHANNEL PROGRAM FOR DIRECT-ACCESS UPDATE, NO TRACK-OVERFLOW (IGG0191Z, IGG01923) FIXED BLOCKED STANDARD OR FIXED STANDARD

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
* WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR A	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	WRT DATA[1] (WRITE CHECK)[7]	BUFFER	C[6]S	DCBBLKSI[2]
CCW5	SET SECTOR	SECTOR A	C	1
CCW6	SCH ID EQ	IOBSEEK+3	C	5
CCW7	TIC	*-8		
CCW8	RD DATA[3]		C[4]SK	DCBBLKSI[2]
CCW9	SEEK HEAD/NOP[5]	CCW17+1	C	6
CCW10	SET SECTOR	SECTOR1	C	1
CCW11	SCH ID EQ	CCW17+3	C	5
CCW12	TIC	*-8		
CCW13	TIC	CCW14		
CCW14	RD COUNT	NEXT CCW17	CS	5
CCW15	RD DATA	BUFFER[2]	CS	DCBBLKSI[2]
CCW16	RD SECTOR	SECTOR2		1
CCW17	MBBCCHHR			

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip

1. If key is specified, WRT KD.
2. For QSAM only.
3. If key is specified, READ KD.
4. Command chain is off for BSAM.
5. CCW9 is present for QSAM only. Op code is SEEK HEAD for cylinder allocation and NOP for track allocation.
6. Command chain is off for BSAM if write-check is not specified.
7. CCWs 5-8 are present for write-check.

APPENDIX D: CHAINED SCHEDULING CHANNEL PROGRAMS

CHANNEL PROGRAM FOR MAIN IOB, CHAINED SCHEDULING (IGG0191K)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	TIC	ICB CH PGM [1]		
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	TIC	ICB CH PGM [1]		

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip
P = PCI

1. TIC goes to the first ICB channel program in the chain.

CHANNEL PROGRAM FOR CHAINED SCHEDULING, INPUT, DIRECT ACCESS (IGG0191K)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	M/T RD COUNT	NEXT ICBSEEK+3	CSP	5
CCW2	M/T RD DATA[1]	BUFFER	CS[2]	BUF LENGTH
CCW3	NOP	CCW1 IN NEXT ICB	S	1
WITH ROTATIONAL POSITION SENSING				
CCW1	M/T RD COUNT	NEXT ICBSEEK+3	CP	8
CCW2	M/T RD DATA[1]	BUFFER	CS	BUF LENGTH
CCW3	TIC (HEX'88')	*+8[3]	S	
CCW4	RD SECTOR	SECTOR 2		1

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip
P = PCI

1. If keys are specified, the command code is READ KD.
2. SLI bit is on for format-U or format-V records.
3. Appendage may change to CCW1 in the next ICB.

CHANNEL PROGRAM FOR CHAINED SCHEDULING, DIRECT ACCESS, OUTPUT, NO WRITE-CHECK (IGG0191W)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	M/T SCH ID EQ	ICBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	WRT CKD	CCW6	DSP	8
CCW4	00	BUFFER	CS	BUF LENGTH
CCW5	NOP	CCW1 or 3 in NEXT ICB	S	1
CCW6	CCHHRKDD			
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR = 0	C	1
CCW2	M/T SCH ID EQ	ICBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	WRT CKD	CCW8	DSP	8
CCW5	00	BUFFER	CS	BUF LENGTH
CCW6	TIC (X'88')	* +8[1]	S	
CCW7	RD SECTOR	SECTOR 2		1
CCW8	CCHHRKDD			

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip
P = PCI

1. Appendage may change to CCW1 or 4 in the next ICB.

CHANNEL PROGRAM FOR CHAINED SCHEDULING, INOUT, OUTIN, ICB WITHOUT WRITE-CHECK (IGG0191X)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITH RECORD READY				
CCW1	M/T RD CNT	NEXT ICBSEEK+3	CSP	5
CCW2	M/T RD DATA		CS [1]	
CCW3	TIC (X'88')	* +8 [2]	S	
CCW4	READ SECTOR	SECTOR2		1
CCW5	M/T SCH ID EQ	ICBSEEK+3	CS	5
CCW6	TIC	*-8		
CCW7	WRT CKD	CCW11	DSP	8
CCW8			CS	
CCW9	TIC ('88')	* +8 [2]	S	
CCW10	READ SECTOR	SECTOR2		1
CCW11	CCHHRKDD			

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip
P = PCI

1. SLI bit is on for format-U and format-V records.
2. Appendage may change to the next ICB CCW.

CHANNEL PROGRAM FOR CHAINED SCHEDULING, INOUT, OUTIN, ICB WITH WRITE-CHECK (IGG0191X)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITH ROTATIONAL POSITION SENSING				
CCW1	M/T RD CNT	NEXT ICBSEEK+3	CSP	5
CCW2	M/T RD DATA		CS	
CCW3	TIC ('88')	*+8[1]	S	
CCW4	READ SECTOR	SECTOR2		1
CCW5	M/T SCH ID EQ	ICBSEEK+3	CS	5
CCW6	TIC	*-8		
CCW7	WRT CKD	CCW15	DSP	8
CCW8			CS	
CCW9	READ SECTOR	SECTOR2	C	1
CCW10	SET SECTOR	SECTOR2	C	1
CCW11	SCH ID EQ	CCW15	CS	5
CCW12	TIC	*-8		
CCW13	M/T RD DATA		CSK	
CCW14	NOP	NEXT ICB CCW7	S	1
CCW15	CCHHRKDD			

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip
P = PCI

1. Appendage may change to next ICB CCW.

CHANNEL PROGRAM FOR CHAINED SCHEDULING, INOUT, OUTIN, ICB (IGG0191R)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	M/T RD CNT	NEXT ICBSEEK+3	CSP	5
CCW2	M/T RD DATA		CS[1]	1
CCW3	NOP	NEXT ICB CCW2	S	
CCW4	M/T SCH ID EQ	ICBSEEK+3	CS	5
CCW5	TIC	*-8		
CCW6	WRT CKD	CCW12	DSP	8
CCW7	(WRITE CHECK[2])		CS	
CCW8	SCH ID EQ	CCW12	CS	5
CCW9	TIC	*-8		
CCW10	M/T RD DATA		CSK	
CCW11	NOP	NEXT ICB CCW6	S	1
CCW12	CCHHRKDD			

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip
P = PCI

1. SLI bit is on for format-U and format-V records.
2. If there is no request for a write-check, CCWs 8-10 are omitted and CCW7 has no command chaining.

CHANNEL PROGRAM FOR CHAINED SCHEDULING, MAIN IOB, INOUT, OUTIN (IGG0191X)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	TIC	ICB CH PGM [1]		
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	TIC	ICB CH PGM [1]		

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip
P = PCI

1. TIC goes to the first ICB channel program in the chain.

APPENDIX E: BSAM (BDAM CREATE) CHANNEL PROGRAMS

CHANNEL PROGRAM FOR ERASE CCWs FOR BSAM LOAD MODE, TRACK-OVERFLOW (IGG0191M)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
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				
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

D = Data Chain
C = Command Chain
S = SLI
K = Skip

1. Address of CCW1 is stored in DCBEOBW

CHANNEL PROGRAM FOR BSAM LOAD MODE, TRACK-OVERFLOW (IGG0191M)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
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	K_L+D_L

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip

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 R0 is to be written on this track.
3. CCWs 10-18 are repeated as many times as are needed to write all segments.

CHANNEL PROGRAM FOR CREATE BDAM (IGG0199L)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
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_1+BLKSIZE
CCW5	SCH ID EQ	IOBDNRCF	C	5
CCW6	TIC	*-8		
CCW7	RD KD		CSK	256
CCW8	SCH ID EQ	IOBR0CNT	C	5
CCW9	TIC	*-8		
CCW10	WRT DATA	IOBR0DAT	C	8
CCW11	SCH ID EQ[2]	IOBR0CNT	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

1. CCWs 5-7 are omitted if write-check is not specified.
2. CCWs 11-13 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)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITH ROTATIONAL POSITION SENSING				
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	K_L+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	IOBROCNT	C	5
CCW13	TIC	*-8		
CCW14	WRT DATA	IOBRODAT	C	8
CCW15	SET SECTOR [3]	SECTOR2	C	1
CCW16	SCH ID EQ	IOBROCNT	C	5
CCW17	TIC	*-8		
CCW18	RD DATA		CSK	1
CCW19	ERASE[2]	*	S	8

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip

1. CCWs 7-10 are omitted if write-check is not specified.
2. CCW19 is generated for format-U and format-V records only.
3. CCWs 15-18 are always generated for format-V and format-U records or if write-check is specified.

CHANNEL PROGRAM FOR BSAM LOAD MODE, TRACK OVERFLOW (IGG0199M)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITH ROTATIONAL POSITION SENSING				
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	K_L+D_L

FLAGS

D = Data Chain
C = Command Chain
S = SLI
K = Skip

1. CCWs 11, 12, 22, 23, and 25-29 are omitted if verify is not specified.
2. The TIC/NOP at CCW6 and CCW17 is set to NOP if R0 is to be written on this track.

INDEX

For additional information about any subject listed in this index, refer to the publications that are listed under the same subject in *OS/VS2 Master Index of Logic*, GY28-0694.

A

ABEND codes cross-reference table 259-262
abnormal-end appendages 108-111
access conditions for selecting modules (*see* module selector)
access method options (*see* module selector)
access method save area for user totaling 254
address conversion routines
  full-to-relative address 192
  relative-to-full address 191-192
ANS control character (*see* control character)
appendages
  introduction to 87-88
  module selector (Figure 15) 89
  types
    abnormal-end 108-111
    channel-end 94-106
    end-of-extent 88-93
    PCI 106-108
    SIO 93-94
appendixes (*see* Contents)
ASCII block prefix 17
associated data set processing
  (*see also* 3505/3525)
  FOB modules (Figure 7) 55
asynchronous-error-processing routines
  track overflow 85-86
  3211 Printer 86-87

B

backspace
  BSP routine 187-188
basic direct-access method, BDAM (*see* BDAM-create)
basic partitioned access method (*see* BPAM)
basic sequential access method (*see* BSAM)
BDAM-create (WRITE-load)
  channel programs (Appendix E) 289-293
  Check routines 126-127
  stage 2 Open executors 153-163
  Write modules 114-123
BDW (block descriptor word) 50-52
BLDL or FIND routines 190-192
  in TRR 201
block descriptor word (BDW) 50-52
block prefix, ASCII 17
blocked records
  Get routines
    simple-buffering 17-32
    update-mode (PUTX) 33-40
  Put routines
    simple-buffering 40-52
    update-mode (PUTX) 52-53
BPAM
  description of routines 134
  flow of control (Diagram G) 219
  introduction to 15,134
  relation to BSAM routines 15,134
  relation to processing program 15,134
  residence of 134
  routines for
    BLDL 190-191
    convert MBBCCHRR 139,191-192
    convert TTR 139,191-192
    FIND 190-192
    STOW 188-190
BSAM
  control blocks 256
  flow of control (Diagram G) 219
  introduction to 15,114
  module selector for
    appendages (Figure 15) 89
    Check (Figure 19) 124
    Control (Figure 20) 129
    overview (Diagram A) 207
    Read (Figure 18) 115
    Write (Figure 18) 115
  routines
    appendages 87-111
    Check 123-127
    Control 128-134
    end-of-block 53-77
    Read 114-123
    synchronizing-and-error processing 77-87
    Write 114-123
BSAM (BDAM-create) channel programs
  (Appendix E) 289-293
BSAM/QSAM channel programs
  (Appendix B) 265-274
BSP
  BSAM overview (Diagram A) 207
  in TRR 204
  routine 187-188
buffer alignment 178-179
buffer empty (Get routines)
  simple-buffering 17-32
  update-mode 33-40
buffer pool management
  BUILD routine 179-181
  FREEBUF (macro expansion) 180
  FREEPOOL (macro expansion) 181
  GETBUF (macro expansion) 180
  GETPOOL routine 178-179
  introduction 15,178
buffer ready for emptying (Put routines)
  simple buffering 40-52
  update mode, PUTX 36-37,52-53
buffering techniques
  Get routines 17-40
  Put routines 40-53
BUILD
  buffer pool management routine 179-181
  common access method routine
    (Diagram A) 207
BUILDRCD
  buffer pool management routine 181
  QSAM overview (Diagram A) 207

C

card punch, 3525 (*see* 3505/3525)
card reader
  (*see also* 3505/3525)
  Get routines 23
chained channel-program scheduling
  appendages
    end-of-extent 92
    PCI, channel end, abnormal 106-108
  channel programs (Appendix D) 281-287
  end-of-block routines 64-74
  IOB prefix 65
  joining
    description of end-of-block routines 64-74
    introduction to 64-65
  Note/Point routines 132-134
  parting (disconnecting) 106-108
  stage 2 Open executors 148-163
  stage 3 Open executors 163-171
chained scheduling, channel programs
  (Appendix D) 281-287
channel-end appendages 94-106
channel programs
  BDAM create (Appendix E) 289-293
  BSAM/QSAM (Appendix B) 265-274
  chained scheduling (Appendix D) 281-287
  update (Appendix C) 275-279
character conversion (*see* paper tape character conversion routines)
CHECK macro instruction
  BSAM/BPAM (Diagram C) 211
  Check modules 123-127
Check routines
  BSAM/BPAM (Diagram C) 211
  descriptions 123-127
Checkpoint records, DOS (*see* OS/DOS tape compatibility)
Close executors 172-176
CLOSE macro instruction
  SAM overview (Diagram A) 207
CNTRL macro instruction
  BSAM control routines 128-134
  QSAM control routines 111-113
code conversion routines
  descriptions (Appendix A) 263-264
common routines
  appendages 87-111
  buffer pool management 178-183
  executors 135-178
  SAM overview (Diagram A) 207
compatibility interface control block, JES 258
control blocks, relation of
  BSAM 256
  compatibility interface (JCICB) 258
  QSAM 255
control character, end-of-block routines
  chained scheduling 64-74
  normal scheduling 54-64
control modules
  selected and loaded by Open executor (Figure 20) 129
Control routines
  BSAM 128-134
  QSAM 111-113
convert full-to-relative address routine 192
convert record number to sector value routine 192
convert relative-to-full address routine 191-192
create-BDAM (*see* BDAM-create)
cross-reference table, ABEND codes 259-262
CSECT names (as listed in the directory) 237-241

D

data areas
  access method save area
    for user totaling 254
  BSAM control blocks 256-257
  JES Compatibility Interface Control Block 258
  QSAM control blocks 255
data operating mode
  Get module (Figure 1) 19,31-32
  Put module (Figure 6) 42,49-52
data protection image, DPI
  (*see also* 3505/3525)
  EOB modules (Figure 7) 55
DCB relocation to protected work area 135-136
decision tables (*see* module selector)
DEVTYPE SVC routine 185
  in TRR 202
diagrams 207-235
directory module names 237-241
DMABCOND macro 136
DOS embedded checkpoint records (*see* OS/DOS tape compatibility)
DPI, data protection image
  (*see also* 3505/3525)
  EOB modules (Figure 7) 55
dummy data set routine 135

E

embedded checkpoint records, DOS (*see* OS/DOS tape compatibility)
empty buffer
  Get routines
    simple-buffering 17-32
    update-mode 33-40
  Put routines
    simple-buffering 40-52
    update-mode, PUTX 36-37
end-of-block condition (*see* end-of-block routines)
end-of-block routines, QSAM/BSAM
  chained channel-program scheduling 64-74
  flow of control (Diagram F) 217
  INOUT or OUTIN 53-64
  Put routines 40-53
  track overflow 74-77
end-of-extent appendages 88-93
EODAD routine
  dummy data set 135
EOV (end-of-volume) routine
  BSAM Flow of Control (Diagram I) 223
erase routine, track overflow 186
error option implementation
  SYNAD routines 192-197
  synchronizing and error processing routines 77-87
EXCP processing with the 3800 printer 58,59
execution of channel programs
  scheduled by chaining (PCI appendage) 106-108

executors, SAM
control sequence (Figure 23) 135
Close 172-176
flow of control for Open (Diagram E) 215
introduction to 136
Open 136-171
relation to Open/Close/EOV support 135
executors, SETPRT 200-200.3
return codes 262-262.2
selection of 197.1
work area 243

F

FIND macro instruction
C option (macro expansion) 190
D option routine 191
flow of control, diagrams for
BPAM routines (Diagram G) 219
BSAM routines (Diagram G) 219
EOV executors
BSAM (Diagram I) 223
QSAM (Diagram H) 221
FEOV executor, QSAM (Diagram J) 225
JES Compatibility Interface routines
(Diagram M) 231
QSAM routines (Diagram F) 217
SAM Open executors (Diagram E) 215
force close executors 177-178
forward space
control module 131-132
FREEBUF macro instruction
BSAM/BPAM (Diagram A) 207
macro expansion 180
FREEPOOL macro instruction
macro expansion 181
SAM overview (Diagram A) 207
full buffer
Get routines
simple-buffering 17-32
update-mode 33-40
Put routines (*see* buffer ready for emptying)

G

GET macro instruction
Get routines 17-40
introduction to Get routines 17
Get routines
buffering techniques 17-19
introduction to 17
simple-buffering 17-32
update mode 33-40
GETBUF macro instruction
BSAM/BPAM (Diagram A) 207
macro expansion 180
GETPOOL
buffer pool management routine 178-179
common access method routine (Diagram A) 207

I

IGGSPW—SETPRT work area 243
IHASPP—SETPRT parameter list 254
IMGLIB SVC routine 185
in TRR 205-206
INOUT mode
end-of-block routines 53-64
stage 2 open executors 148-163
input processing routine, QSAM 32-33
I/O interruption
BPAM flow of control (Diagram G) 219
QSAM flow of control (Diagram F) 217
SAM overview (Diagram A) 207
IOB (input/output block) SAM prefixes
comparison for normal end chained-scheduling 65
description 64-65

J

JES Compatibility Interface Control
(*see also* SAM-SI)
BSAM processing module (SYSOUT) 127
Check module (SYSIN) 123-124
Close processing (Diagram L) 229
Control block (CICB) 258
Open processing (Diagram K) 227
Overview (Diagram M) 231
QSAM processing module 28

L

load-BDAM (*see* BDAM-create)
logical record interface
Get module 27
Put module 46-48

M

macro expansion
FIND (C option) 190
FREEBUF 180
FREEPOOL 181
GETBUF 180
PRTOV 111-113
MBBCCHRR, convert address routine 192
module name directory 237-241
module selector
appendages (Figure 15) 89
Check modules (Figure 19) 124
Close executors (Figure 27) 172
Control routines
BSAM (Figure 20) 129
QSAM/BSAM (Figure 16) 112
end-of-block modules
chained scheduling (Figure 9) 66
ordinary (Figure 7) 55
track overflow (Figure 12) 75
error processing modules (Figure 14) 85
Get modules
simple-buffering (Figure 1) 19
update-mode (Figure 5) 36
Open executors
stage 1 (Figure 24) 138
stage 2 (Figure 25) 149
stage 3 (Figure 26) 165
Put modules
simple-buffering (Figure 6) 42
update-mode, PUTX (Figure 5) 36
Read modules (Figure 18) 115
synchronizing and error processing modules
(Figure 13) 79
Write-modules (Figure 18) 115

module selector tables (*see* module selector)
module type
  (as listed in the directory) 237-241

N

name (module) directory 237-241
new buffer (*see* empty buffer Put routines; full buffer Get routines)
new buffer segment (Put routines)
  simple-buffering 40-52
  update-mode (PUTX) 33-40
new programming support
  3800 printing subsystem 11
next record (Get routines)
  simple-buffering 17-32
  update-mode 33-40
non-rotational position sensing indicator (*see* rotational position sensing, Open executors)
NOTE macro instruction
  BSAM/BPAM overview (Diagram A) 207
  BSAM control routines 128-134
Note/Point routines
  BSAM control routines 128-134
  chained scheduling 133
  normal scheduling 128-132
  track overflow 132
  update mode 132-133

O

OMR (optical mark read) (*see* 3505/3525)
Open executors, SAM
  for the 3800 printer 144,145
  introduction to 136
  stage 1 136-148
  stage 2 148-163
  stage 3 163-171
OPEN macro instruction
  general flow (Diagram D) 213
  SAM overview (Diagram A) 207
OPTCD=J and the 3800 printer 58,59,70
OPTCD=Z (*see* search direct)
optical mark read (*see* 3505/3525)
optional, access method (*see* module selector)
OS/DOS tape compatibility
  appendages 102-104
  BSAM control routines 130-133
  Get routines 17
  synchronizing-and-error processing routines 81
OUTIN mode
  end-of-block routines 53-64
  stage 2 open executors 148-163
overview, SAM (Diagram A) 207

P

paper tape channel-end appendage 100-101
paper tape character conversion routines
  Check routine 125
  Get routine 26
  Read routine 116-117
  stage 2 open executor 151
  stage 3 open executor 166
  synchronizing and error processing routine 26
paper tape code conversion modules
  (Appendix A) 263-264
parallel data address block 32
parallel input processing routine 32
parting chained channel-programs, appendage 106-107
PCI appendages 106-108
POINT macro instruction
  BSAM/BPAM overview (Diagram A) 207
  BSAM control routines 128-134
Point routines (*see* Note/Point routines)
priming input buffers
  introduction to
    simple-buffering 17
    update-mode 33
  stage 3 open executor 163
printer (*see* 3203 printer, 3211 Printer, 1403 Printer, or 3505/3525)
printer overflow macro expansion
  (PRTOV) 113
problem determination 183-184
processing program
  relation to SAM routines (Diagram A) 207
  using BPAM routines 134
program controlled interruption, appendages 106-108
program organization, diagrams for
  BPAM routines (Diagram C) 211
  BSAM routines (Diagram C) 211
  Open executors (Diagram D) 213
  QSAM routines (Diagram B) 209
  SAM routines (Diagram A) 207
  SAM-SI (Diagram M) 231
protected work area, DCB relocation to 135-136
PRTOV macro instruction
  appendage 94
  BSAM 128
  end-of-block routines 58-59
  QSAM 111-113
PUT macro instruction
  introduction to Put routines 40
  Put routines 40-53
Put routines
  simple-buffering 40-52
  update-mode (PUTX) 52-53
PUTX macro instruction
  overview (Diagrams A,B) 207,209
  Put routines 40-53
PUTX routine
  simple-buffering 40-52
  update mode
    Get routine 34-37
    PUTX 52-53

Q

QSAM
  control blocks 255
  control routines 111
  flow of control (Diagram E) 215
  introduction to 15,17
  module selector for
    simple-buffering, Get (Figure 1) 19
    simple-buffering, Put (Figure 6) 42
    update-mode, Get (Figure 5) 36
    update-mode, PUTX (Figure 5) 36
  overview (Diagram A) 207

routines
appendages 87-111
Control 111-113
end-of-block 53-74
Get 17-40
Put 40-53
synchronizing-and-error-processing routines 77-87
queued sequential access method (*see* QSAM)

R

RCE, read column eliminate (*see* 3505/3525)
read column eliminate (*see* 3505/3525)
READ macro instruction
BSAM/BPAM (Diagram C) 211
Read routines (Figure 18) 114-123
Read routines
BSAM/BPAM (Diagram C) 211
descriptions (Figure 18) 114-123
readback mode, Get routines (Figure 1) 19,23-25
record descriptor word (RDW) 29,31
record number conversion to sector value 192
RELSE macro instruction
Get routines 18-40
overview (Diagram A) 207
RELSE routines
description (Get routines) 18
simple-buffering 17-32
update mode 33-40
return codes from SETPRT executors 262-262.2
rotational position sensing (RPS)
appendages
channel-end 94-95,98-99
end-of-extent 90
PCI 107
SIO 94
channel programs (Appendixes B,C,D,E) 265-293
Get routines
update mode 36
Open executors
introduction 136
Read/Write routines 114-123
stage 2 148-163
stage 3 163
RPS (*see* rotational position sensing)

S

SAM
common routines
appendages 87-111
buffer pool management 178-183
executors 135
effect of BLDLTAB
force close executor 177-178
introduction to 15
overview (Diagram A) 207
Register Saving Convention 262.2
SAM-SI (SAM subsystem interface)
QSAM
introduction to 15
force close executor 177
Get routine 28
Put routine 48
synchronizing-and-error-processing routine 81,84
SAM
Check routines 123-125
Read, Write routines 122-123
scheduling (*see* chained channel-program scheduling; end-of-block routines)
SDW (*see* segment descriptor word)
search direct (OPTCD=Z)
appendages (Figure 15) 89
channel programs (Appendix B) 265-273
stage 1 Open executors 138
stage 2 Open executors 150
stage 3 Open executors 168-171
search-previous auxiliary storage
addressing 34
seek address in QSAM update mode 34-35
segment descriptor word (SDW)
Get routines 27-32,38
Put routines 50-52
sequential access methods (*see* SAM)
sequential access method executors (*see* executors, SAM)
SETDEV routine 200
SETPRT
executors 200-200.3
messages 262-262.4
return codes 262-262.4
selector 197.1
in TRR 204-205
parameter list 254
QSAM/BSAM overview (Diagram A) 207
routines 197-200
work areas 244-244.4
simple buffering
Get routines 17-32
Put routines 40-52
SIO appendages 93-94
space magnetic tape
BSP routine 187-188
Control routine 131
spanned records
Get routines 27-32
Put routines 48-52
stage 1 Open executors
descriptions 136-148
flow of control (Diagram E) 215
module selector (Figure 24) 138
stage 2 Open executors
descriptions 148-163
flow of control (Diagram E) 215
module selector (Figure 25) 149
stage 3 Open executors
descriptions 163-171
flow of control (Diagram E) 215
module selector (Figure 26) 165
start I/O (SIO) appendages 93-94
STOW routines 188-190
in TRR 201
SVC routines
descriptions 184-200
directory entries 237
SVRB extended save area 244.4
SYNAD/EOV executor
flow of control (overview)
BSAM (Diagram I) 223
QSAM (Diagram H) 221
SYNAD routine, FEOV executor
QSAM operation for output data set (Diagram J) 225

SYNADAF/SYNADRLS routines 192-197
in TRR 202-204
synchronizing-and-error-processing routines
asynchronous-error-processing 85-87
introduction to 77-78
QSAM (Figure 13) 79-84
track overflow 85-86
3203 or 3211 printer asynchronous-error-processing 86-87

T

tables (*see* module selector)
tape compatibility, OS/DOS
appendages 102-104
Control routines 130-133
Get routines 17
synchronizing-and-error-processing routines 81
task recovery routines (TRR)
SVC 18—BLDL or FIND 201
SVC 21—STOW 201
SVC 24—DEVTYPE 202
SVC 25—track balance, track overflow erase 202
SVC 68—SYNADAF/SYNADRLS 202-204
SVC 69—BSP 204
SVC 81—SETPRT 204-205
SVC105—IMGLIB 205-206
track balance routine 186
in TRR 202
track erase routine 186
track overflow
abnormal end appendage 108-111
create-BDAM write routine 121-122
end-of-block routine 74-77
Erase routine 186
error processing routine 85-86
in TRR 202
introduction to 74
stage 2 Open executors 151,153-157
stage 3 Open executor 166,167
translate routine, IGC0010C (Appendix A) 264
TRKCALC macro 192
TRUNC macro instruction
overview (Diagram A) 207
Put routines 40-53
TRUNC routines
description (Put routines) 43-53
simple-buffering 40-52
TTR, convert address routine 191-192

U

UCS feature, printer
stage 1 Open executors 136-147
unblocked records
Get routines
simple-buffering 17-19
update mode 33
Put routines
simple-buffering 40-42
universal character set (*see* UCS feature, printer)
update channel programs (Appendix C) 275-279
update mode
appendages
end-of-extent 88-93
SIO 93-94
Check routine 126
Get routines 33-40
Note/Point routine 132-133
PUTX routine 52-53
Read/Write routine 117
schedule buffer (empty-and-refill or refill only) 33-35
stage 2 Open executors 154-155,158
stage 3 Open executors 164-166
synchronizing routine 79-81
user totaling facility
access method save area 254
end-of-block modules 74-77
stage 1 Open executors 137-145

W

WRITE-load (*see* BDAM-create)
WRITE macro instruction
BSAM/BPAM (Diagram C) 211
Write routines 114-123
Write routines
BSAM/BPAM (Diagram C) 211
descriptions 114-123

X

XLATE macro instruction (Appendix A) 264

123

1403 Printer
Open executor, stage 1 136-148
2520/2540 card read punch
consideration of DCBBUFNO field 144
3203 or 3211 Printer
appendage 108,111-112
asynchronous-error-processing module 86
Open executor, stage 1 136-148
Open executor, stage 3 167
synchronizing module 82
3505/3525 (card reader, card punch)
Close executors (Figure 27) 172
control routine 112-113
EOB modules 56-62
line control (Figure 16) 112
Open executor, stage 1 (Figure 24) 136-148
Open executor, stage 2 (Figure 25) 149,161
print (EOB module) 60-61
3525 card punch (*see* 3505/3525)
3800 printing subsystem
and EXCP 58,59
and OPTCD=J 58,59,70
area in SETPRT work area 245
new programming support 11
3886 Optical Character Reader
Open executors, stage 1 (Figure 24) 136-148

IBM Technical Newsletter

This Newsletter No. SN26-0956
Date 30 November 1979

Base Publication No. SY26-3832-1
File No. S370-30

Prerequisite Newsletters SN26-0917
SN26-0931
SN26-0934

OS/VS2 SAM Logic

This technical newsletter, a part of Release 3.8 of OS/VS2, provides replacement pages for the subject publication. These replacement pages remain in effect for any subsequent releases unless specifically altered. Pages to be inserted and/or removed are:

title page, edition notice
3–11 (12-14 removed and not replaced)
75, 76
103, 104
141–142.1
157–160
181–186
197–200.6 (200.5 and 200.6 added; text rearranged)
241–246 (244.5 removed and not replaced)
253–254.2
261–262.4 (262.3 and 262.4 added)
299, 300

Each technical change is marked by a vertical bar to the left of the change.

Summary of Amendments

Changes included in this newsletter are summarized under "Summary of Amendments" following the list of diagrams.

Note: *Please file this cover letter at the back of the publication to provide a record of changes.*

IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California 95150

Printed in U.S.A.

IBM Technical Newsletter

This Newsletter No. SN26-0931
Date July 2, 1979

Base Publication No. SY26-3832-1
File No. S370-30

Prerequisite Newsletters SN26-0917
SN26-0934

OS/VS2 SAM Logic

This technical newsletter provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent releases unless specifically altered. Pages to be inserted and/or removed are:

Cover, Edition Notice
11–13
81, 82
141–142.1
147, 148
191–192.1 (192.1 added)
299, 300

Each technical change is marked by a vertical bar to the left of the change.

Summary of Amendments

Changes included in this newsletter are summarized under "Summary of Amendments" following the list of figures.

Note: Please file this cover letter at the back of the publication to provide a record of changes.

IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California 95150

Printed in U.S.A.

IBM® **Technical Newsletter**

This Newsletter No. SN26-0934
Date March 30, 1979

Base Publication No. SY26-3832-1
File No. S370-30

Prerequisite Newsletters SN26-0917

OS/VS2 SAM Logic

This technical newsletter, a part of Release 3.8 of OS/VS2, provides replacement pages for the subject publication. These replacement pages remain in effect for any subsequent releases unless specifically altered. Pages to be inserted and/or removed are:

Cover, Edition Notice
11, 12
55–58.1 (58.1 added)
63–64.1 (64.1 added)
67–78.2 (78.1 and 78.2 added)
137–142.1 (142.1 added)
251, 252
259, 260

Each technical change is marked by a vertical bar to the left of the change.

Summary of Amendments

Changes included in this newsletter are summarized under "Summary of Amendments" following the list of diagrams.

Note: *Please file this cover letter at the back of the publication to provide a record of changes.*

IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California 95150

Printed in U.S.A.

IBM Technical Newsletter

This Newsletter No. SN26-0917
Date August 31, 1978

Base Publication No. SY26-3832-1
File No. S370-30

Prerequisite Newsletters None

OS/VS2 SAM Logic

This technical newsletter, a part of Release 3 of OS/VS2 MVS, incorporates and replaces all previous SU information in this publication. Please replace all identically numbered pages. These replacement pages remain in effect for subsequent releases unless specifically altered. Pages to be inserted and removed are:

Cover, Edition Notice
3 – 13
57 – 60.1 (60.1 added)
69 – 70.1 (70.1 added)
137 – 138.1 (138.1 added)
143 – 146.1 (146.1 added)
151 – 156
195 – 200.4 (200.1 – 200.4 added)
239 – 246 (244.1-244.5 added)
253 – 254.2 (254.1, 254.2 added)
259 – 262.2 (262.1, 262.2 added)
295 – 300

Each technical change is marked by a vertical line to the left of the change.

Summary of Amendments

Changes are summarized under "Summary of Amendments" following the list of illustrations.

Note: Please file this cover letter at the back of the publication to provide a record of changes.

IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California 95150

Printed in U.S.A.

OS/VS2 SAM Logic (File No. S370-30) Printed in U.S.A. SY26-3832-1

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

OS/VS2 SAM Logic
SY26-3832-1

Reader's Comment Form

Your comments about this publication will help us to improve it for you. Comment in the space below, giving specific page and paragraph references whenever possible. All comments become the property of IBM.

Please do not use this form to ask technical questions about IBM systems and programs or to request copies of publications. Rather, direct such questions or requests to your local IBM representative.

If you would like a reply, please provide your name, job title, and business address (including ZIP code).

Fold on two lines, staple, and mail. No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

Fold and Staple

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
System Development Division
LDF Publishing—Department J04
1501 California Avenue
Palo Alto, California 94304

Fold and Staple

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

OS/VS2 SAM Logic (File No. S370-30) Printed in U.S.A. SY26-3832-1