File No. S360-36
Order No. GY28-6660-9

Program Logic

IBM System/360 Operating System: MVT Job Management, Program Logic Manual,

Program Number 360S-CI-535

OS Release 21

This publication describes the internal logic of the job management routines for the MVT control program of the IBM System/360 Operating System. Included are discussions of input stream processing, work queue management, job initiation and termination, I/O device allocation, system output processing, and the scheduling and execution of operator commands.

This manual is intended for persons involved in program maintenance, or system programmers who are altering the program design. Program logic information is not necessary for the use and operation of the program.

Tenth Edition (March, 1972)

This is a major revision of, and obsoletes, GY28-6660-8. The publication has been modified to reflect the changes described in the "Summary of Amendments." Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

This edition applies to release 21 of IBM System/360 Operating System, and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/360 and System/370 SRL Newsletter, GN20-0360, for the editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Programming Systems Publications, Department D58, PO Box 390, Poughkeepsie, N. Y. 12602. Comments become the property of IBM.

Preface

This book describes the internal logic of the IBM System/360 Operating System job management routines for the MVT control program. It includes discussions of input stream processing, work queue management, job initiation and termination, I/O device allocation, system output processing, and the scheduling and execution of operator commands.

The book is intended for persons involved in program maintenance, or for system programmers responsible for altering the program design.

The introduction to this book describes job management in terms of the system tasks performed. Parts 1-6 describe the functions performed and the flow of control among the routines. The appendixes contain format descriptions of the major work areas and tables used by job management, brief descriptions of job management modules, module cross references, flowcharts, and an abbreviation dictionary.

Users of this book should have a thorough knowledge of IBM System/360 programming. The following publications provide basic background information:

IBM System/360 Operating System:

Job Control Language Reference Manual, GC28-6704

Supervisor Services, GC28-6646

A more complete list of books providing reference and background information is contained in the Bibliography, which precedes the Index to this publication.

Contents

Illustrations

Figures

Charts

Summary of Amendments for GY28-6660-9 OS Release 21

NEW PROGRAMMING FEATURES

Status Display Support

The following task-creating commands have been added to the COMMAND PROCESSING section: DISPLAY PFK; DISPLAY C,K; DISPLAY M (removed from the list of existing-task commands); MONITOR A.
The following existing-task commands have been added to the COMMAND PROCESSING section: MSGRT; STOPMN; CONTROL.

The operands in the following list have been moved from the STOP command to the STOPMN command: DSNAME, SPACE, JOBNAMES, STATUS, SESS. Note: For release 21, the system will continue to recognize these operands as valid for use with the STOP command as well as for the STOPMN command.

M65MP Shared DASD
Changes have been made to some VARY commands for shared direct access storage device support.

Console DUMP Command
A new SVCLIB module has been added to provide for dumping of main storage to a pre-allocated data set, SYS1.DUMP.

VARY with OLTEP
The VARY CONSOLES and VARY ONLINE routine has been modified for use of an online test executive program.

DISPLAY SQA Command
A new SVC 34 command has been added to display the system queue area.

Automatic Volume Recognition
Changes have been made to reflect new devices and volumes and processing modifications in this area of the book.

MISCELLANEOUS CHANGES

SET Command
A new module has been added to the processing routine for this command.

Queue Manager Parameter Area
The QMPA has been slightly modified.

TIME Macro Instruction
Provision has been made for the use of this macro instruction during subtask attaching and allocation housekeeping routines.

Alternate Step Delete
The Alternate Step Delete routine has been deleted.

Master Scheduler Initialization Routine
Sections of this routine have been rewritten.

Reply Processor for Non-MCS
An SVC 34 routine for non-MCS environments has been added.

TCTIOT
The graphic layout of this table has been modified.

Job Cancellation
Changes have been made to reflect job cancellation processing when in the flush/fail mode. See Allocation Housekeeping section.

Dictionary
New abbreviations have been added to the Appendix E dictionary.

Module Cross-Reference
Appendix C has been updated to reflect new and changed module cross-reference information resulting from other changes in the publication.

Dynamic Data Set Allocation
The description of this feature has been removed from the publication. The subject matter may now be found in the publication IBM System/360 Operating System: Terminal Monitor Program and Service Routines Program Logic Manual, GY28-6770.

NEW PUBLICATION FEATURE

Module Descriptive Name-Assembly Name Cross Reference
Appendix F now contains a cross-reference between the module descriptive names and the assembly listing names that are given in Appendix B. Thus, if the descriptive name is known, the corresponding assembly module name may be found. The appendix also contains references to illustrations that contain the module names.

Summary of Amendments for GY28-6660-8 OS Release 20.1

Name of Item	Description	Area of Publication Affected (areas correspond to entries in Table of Contents)
TSO System Management Facilities (SMF)	SMF routines have been changed to provide SMF recording for jobs operating under TSO.	Part 2 Attaching the Subtask Part 6 Task Termination Step termination SMF Processing System Management Facilities The SMF SVC Routine (SVC 83) Transferring Records The SMF Records Appendix A Job Management Record (JMR) Figure 90 System Management Control Area (SMCA) Timing Control Table (TCT) Figure 102
TSO Master Scheduler Support/Shortened CSCB	The Command Scheduling Control Block size is reduced.	Part 2 Terminating Subtasks Part 5 Command Scheduling Task-Creating Commands The START Commands The DISPLAY A Command Appendix A Command Scheduling Control Block Figure 75 Appendix C Appendix E
TSO Master Scheduler Support STAE in the Master Scheduler	SVC 34 has been changed to provide the STAE facility for error recovery in the master command scheduling routines.	Part 5 Command Scheduling Initializing the Command Scheduling Routine Figure 21 Command Execution Task-Creating Commands The START Command Figure 25

(Part 1 of 5)

Name of Item	Description	Area of Publication Affected (areas correspond to entries in Table of Contents)
TSO Master Scheduler Support STAE in the Master Scheduler (Continued)		Existing-Task Commands The STOP DSNAME Command through The STOP STATUS Command Appendicies B, C, and E
TSO Master Scheduler Support/Operator Subcommands	SVC 34 routines have been changed to support TSO operator subcommands.	Introduction Operator Commands Part 5 Command Scheduling Initializing the Command Scheduling Routine Storage and Notification Table 4 Error Message Processing Command Execution Task-Creating Commands The DISPLAY A Command The Syntax Check Routine Existing-Task Commands The Monitor DSNAME Command through the HALT Command The STOP DSNAME Command through the STOP STATUS Command Appendix A Command Scheduling Control Block Figure 75 Master Scheduler Resident Data Area Figure 92 Appendix B IEEVDSP1, IEEVRC, IEEVRCTL, IEE1403D Appendix E
TSO Scheduler LOGON/LOGOFF	The System Task Control and the Initiator routines have been changed to support TSO job processing	Part 3 Initiator Functions Figure 12 Initializing the Initiator Selecting Jobs Looking for Work, through Data Set Integrity Attaching the Subtask, through Terminating Subtasks Job Termination

(Part 2 of 5)

Name of Item	Description	Area of Publication Affected (areas correspond to entries in Table of Contents)
TSO Scheduler LOGON/LOGOFF (Continued)		Part 5 Command Execution Task-Creating Commands The START Commands Figure 25 Appendix A Job Management Record (JMR) through Linkage Control Table Appendix B
TSO Scheduler Commands	The work queues have been changed to include the TSO background reader queue. The Queue Alter routines can now process the background reader queue.	Introduction Operator Commands Part 1 Initializing the Queue Data Set Part 5 Command Execution Task-Creating Commands The Queue Manipulation Commands The Queue Search Control Routine (Module IEESD563) The System Reconfiguration Commands Part 6 The Work Queues Appendix B Appendix E
Reader Device Type	The reader device type and class field of the JMR has been changes to indicate a dummy IEFRDER DD statement	Appendix A Job Management Record
SWITCH Command Processing	SVC 34 has been changed to include processing of the SWITCH command by module IGC1403D	Part 5 Command Scheduling Storage and Notification Table 4 Command Execution Existing-Task Commands The DISPLAY T Command The SWITCH Command Appendix B IEE1403D Appendix C
Dynamic Allocation	Dynamic Allocation routines for TSO have been added.	Appendix F

(Part 3 of 5)

Name of Item	Description	Area of Publication Affected (areas correspond to entries in Table of Contents)
3211 Printer	Support of the 3211 UCS printer as an output device	Part 1 Initializing and Processing Commands Processing a Queue Entry Processing Data Sets The Data Set Processing Subtask Processing System Message Blocks Service Routines The Transition Routine Figure 19 Figure 88 Appendix B IEFSDTTE
OLTEP	Allocation eligibility for units being tested by OLTEP	Part 6 I/O Device Allocation Unit Assignment Demand Allocation Automatic Volume Recognition Common Subroutines Allocation Recovery Lack of Available Devices Appendix E
MODE Command	Use of MODE command for Model 145	Part 5 Figure 21 Command Execution Existing-Task Commands The MODE Command Appendix B IGF29701
Rotational Positional Sensing	Use of a RPS device for work queues	Part 1 Initializing the Queue Data Set Appendix B IEFSD055 Appendix E
Miscellaneous changes and improvements to existing publication	Modifications to system task control routines	Part 5 Figure 25 Table 6.1 Appendix B IEEVRC IEFVSCAN

(Part 4 of 5)

Name of Item	Description	Area of Publication Affected (areas correspond to entries in Table of Contents)
	Modifications to the START Command	Part 5 Command Execution Task-Creating Commands The START Command
	Modifications to Queue routines	Part 5 Command Execution Task-Creating Commands The Queue Search Routine Figure 28 Appendix B IEESD582 IEESD583
	Modifications to queue entry processing	Part 6 The Interpreter Routine Input and Control Operations Queue Entry Processing

(Part 5 of 5)

Summary of Amendments for GY28-6660-6 OS Release 20

Name of Item	Description	Area of Publication Affected (areas correspond to entries in Table of Contents)
Shortened CSCB	A reduction in the size of the command scheduling control block.	Part 3 Terminating Subtasks--Cancel, Timer, and OUTLIM Termination Part 5 Figure 25 Command Execution--Task-Creating Commands: The START Command Appendix A Figure 75 Appendix B IEEVICTL IEEVSMSG Appendix C Appendix E
STAE in the Master Scheduler	A change to SVC 34 to provide the STAE facility for error recovery in the master scheduling and command scheduling routines.	Part 5 Command Scheduling--Initializing the Command Scheduling Routine Figure 21 Command Scheduling--Storage and Notification Command Scheduling--Error Message Processing Command Execution--Task-Creating Commands Command Execution--Existing-Task Commands: The MODIFY Command, The STOP Command Appendix B IEEPDISC IEEVDSP1 IEEVWAIT IEE0003D IEE5103D

(Continued)

Name of Item	Description	Area of Publication Affected (areas correspond to entries in Table of Contents)
STAE in the Master Scheduler (Continued)		IEE5203D IEE5303D Appendix C Appendix E
ASCII	A change to routines that verify tape labels so that they can process American National Standard labels.	Part 4 Initializing and Processing Commands--Service Routines: The Put Routine, Transition Routine Part 6 I/O Device Allocation--Unit Assignment: Automatic Volume Recognition, Processing Requests for Unmounted Volumes Figure 70 Table 15 Appendix A Figure 88 Appendix E
Model 155 Recovery Management Support	The addition of a routine (module IGF29601) to SVC 34 that enables the operator to display status information about the Model 155 via the MODE command.	Part 5 Figure 21 Command Execution--Existing-Task Commands: The MODE Command Figure 32 Appendix B IGF29601 Appendix C Appendix E
Model 165 Recovery Management Support	The addition of a routine (module IGF55301) to SVC 34 that enables the operator to display status information about the Model 165 via the MODE command.	Part 5 Figure 21 Command Execution--Existing-Task Commands: The MODE Command Figure 32 Appendix B IGF55301 Appendix C Appendix E

(Continued)

Name of Item	Description	Area of Publication Affected (areas correspond to entries in Table of Contents)
Time-of-Day Clock	Routines (modules IGC6503D and IGC6603D) that set the TOD clock for System/370 models during SET command processing.	Part 1 The Master Scheduling Task Part 5 Figure 21 Command Execution--Existing-Task Commands: The SET DATE and SET CLOCK Commands Appendix B IEE6503D IEE6603D Appendix C Appendix E
Delayed Volume Verification	A change to I/O device allocation that delays the verification of volumes containing old data sets until the end of allocation processing.	Part 3 Figure 12 Part 6 I/O Device Allocation--Allocation Housekeeping: Protecting UCB Information Figure 66 I/O Device Allocation--Gathering Information: Completing Tables I/O Device Allocation--Unit Assignment: Processing Requests for Unmounted Volumes I/O Device Allocation--Space Assignment: Mounting Volumes, Compressing the TIOT I/O Device Allocation--Common Subroutines: Unsolicited Device End Posting Routine, Lack of Available Devices, Lack of Available Space, Separation Strike-Out I/O Device Allocation--Allocation Exit Appendix A Figure 74 Appendix B IEFSD41Q IEFVPOST

(Continued)

Delayed Volume Verification (Continued)		IEFWD000 IEFWEXTA IEFXT002 IEFXT003 Appendix C Appendix D Chart 22 Chart 23 Chart 24 Appendix E
High-Speed Job Queue Format	Changes to modules IEFORMAT and IEFSD055 so that module IEFSD055 issues the XDAP macro instruction to write the master QCR into the control record area.	Part 1 Initializing the Queue Data Set--Formatting the Queue Data Set
Improvements to Existing Publication	Changes to the SIOT at decimal offsets 47, 48, 55, and 84 to reflect release 20 mapping.	Appendix A Figure 97
	Changes to the SCTX to reflect release 20 mapping.	Appendix A Figure 96
	Clarification of the passing of control from the Attach routine (module IEFSD263) to the TCTIOT Construction routine (module IEFSMFAT).	Part 3 Attaching the Subtask
	Explanation of passing of control from modules IEFVHEB and IEFVHH to module IEFUJV.	Part 6 Figure 55
	Clarification of mounting direct access volumes.	Part 6 I/O Device Allocation--Space Assignment: Mounting Volumes
	Clarification of SMF exits from the reading task.	Part 6 Figure 13
Calculation of Data Set Device Requirements	Additional calculation of device requirements if both unit affinity and volume affinity apply to the data set, and if the device type specified is tape only.	Part 6 I/O Device Allocation--Gathering Information: Calculating device Requirements Table 16

Introduction

To the user of a computing system, the basic unit of work is the job. The user must arrange for the programs that execute that job to be loaded and executed, he must insure that the required I/O devices are available (and that the appropriate data sets and scratch volumes are mounted), and he must oversee the operation and cancel or change the order in which jobs are run as external conditions dictate. These functions are frequently performed by an operator, following the instructions of programmers and installation management.

With the IBM System/360 Operating System however, these functions are performed by Job Management routines. Since the number and types of I/O devices, the names of data sets, program names, and other descriptive information must be communicated to the operating system, Job Management routines read and interpret job control language (JCL) statements. Since it is often desirable to include data in the input stream of JCL statements, and convenient for the system to write standard output data sets, System Input and Output routines have been included. Operator commands, and the routines to execute those commands, have been included to give the operator control over the processing performed by Job Management routines.

At the MVT level, these functions are performed concurrently by Job Management routines executing the reading, initiating, writing, master scheduling, and command execution tasks.

Processing Input Streams

Jobs are presented to the operating system in input streams consisting of JCL statements, system input data records, and operator command statements. Each time the operator issues a START command specifying a reader, a reading task is established to process an input stream. The interpreter, the program that performs the reading task, includes the following functions:

- It reads records from an input stream and the procedure library.
- It scans JCL statements and converts them to internal text.
- It builds tables from the internal text, and creates input queue entries from the tables.
- It places messages from the operating system to the programmer in a system output queue entry (which, however, is not completed or enqueued until the last step of the job has terminated).
- It assigns space in output queue entries for pointers to system output data sets.
- It writes system input data records to an intermediate, direct access device, and places pointers to a job's system input data sets in its input queue entry.
- It enqueues the job's input queue entry at the priority specified for the job.

Each reading task in the system may be performed concurrently with other reading tasks, as well as with other system tasks and job steps. Performance of a reading task is terminated when the operator issues a STOP command, or when the associated input stream is exhausted.

Initiating Job Steps

Each time the operator issues a START command specifying an initiator, an initiating task is established to schedule the execution of job steps. The initiator is the program that executes an initiating task; it selects the highest priority job from the first of the input queues associated with it that contains entries. If there are no entries available in any of its queues, the initiator enters the wait state until an entry is enqueued or a STOP or MODIFY command is issued.

When a queue entry becomes available, the initiator dequeues it and insures the integrity of data sets for which a nontemporary name has been specified. It selects the steps of the job (in the order in which the EXEC statements appeared), and for each step, performs the following functions:

- It obtains a region of main storage for the step.

- It uses the I/O Device Allocation routine to check EXEC statement condition codes, to allocate I/O devices, to obtain space on direct access devices, to issue mounting messages, to construct the job step TIOT, and to add messages to the job's message class output queue entry.

- It passes information to the supervisor, and passes control to the first program of the job step via the ATTACH macro instruction, then enters the wait state while the step is being executed.

- When the job step has been executed, the initiator uses the Termination routine to check JOB statement condition codes, to execute the User's Accounting routine, to direct the disposition of the data sets referred to during execution of the job step, to free the I/O devices allocated to the job step, and to add messages (and pointers to system output data sets) to the job's output queue entries. If the step is the last step of the job, the Termination routine also enqueues the job's output queue entries.

Each initiating task may be performed concurrently with other initiating tasks and job steps, as well as with other system tasks. However, only one job at a time, and (within a job) only one step at a time is executed as a result of the performance of one initiating task. Thus, the number of job steps that can be executed concurrently is equal to the number of initiating tasks established in the system.

The performance of an initiating task is interrupted during the execution of its subtask (the job step), and when there are no entries in the input queue. Performance of the task is terminated when the operator issues a STOP command.

Writing System Output

System output consists of messages from the operating system to the programmer, and of data sets designated by the programmer as system output data sets. The messages, and in many cases pointers to the data sets, are placed in output queue entries by routines performing the reading and initiating tasks.

System output is divided into classes (as specified by the programmer and the installation). There is an output queue corresponding to each of the 36 available classes. Messages are placed in the message class; data sets may be in the message class, or they may be placed in other classes. There might, for example, be one class for output to a printer. Another class might be for punched output, and a third might be for output to be written on tape for later printing.

There are two ways of having system output written on the appropriate output device:

- The operator may use START commands to initiate direct system output (DSO) processing for one or more classes. He may change the set of classes via MODIFY commands, via STOP commands, or by issuing additional START commands.

- The operator may use START commands to establish writing tasks. He may change the set of classes via MODIFY commands, via STOP commands, or by issuing additional START commands.

If DSO processing has been started, a job having system output in a DSO class writes data directly on the appropriate device. System output in other classes is written on a direct access device for later processing by a system output writer.

If the DSO processing is specified for the message class, the initiator writes messages concerning the job on the appropriate device. Otherwise, it adds the messages to the job's message class output queue entry for later processing by a system output writer.

When a job terminates, the output queue entries that contain messages or pointers to data sets are enqueued. Such messages and data sets must be processed by system output writers.

When the operator issues a START or MODIFY command, to establish or change a writing task, the command specifies the queues from which the writer selects entries, and the writer processes as follows:

- The queues are listed in the order specified in the command; the writer dequeues the highest priority entry from the first queue in its list that contains entries. If there are no entries available in any of its queues, the writer enters the wait state until an entry is enqueued.

- A queue entry may contain messages, pointers to data sets, or both; the writer reads each record in the entry into main storage and determines whether the record contains messages or a pointer to a data set.

- If the record contains messages, the messages are written to the output device, and another record is obtained from the queue entry. If the record points to a data set, the writer opens the data set, attaches a subtask to write the data set records to the output device, then closes and deletes the data set before obtaining another queue entry record.
- When the last record in a queue entry has been processed, the writer deletes the entry before dequeueing another entry.

Each writing task may be performed concurrently with the performance of other writing tasks, as well as with job step tasks and with other system tasks. The performance of the writing task is interrupted during the performance of its subtask, and when there are no output queue entries of the classes in the writer's list. Performance of a writing task is terminated only when the operator issues a STOP command; performance of the subtask may be terminated with a CANCEL command.

Initializing and Establishing System Tasks

The master scheduling task is established when the system is loaded. Routines performing this task initialize the nucleus, console communications, and the system log (see the publications IBM System/360 Operating System: IPL/NIP, Program Logic Manual, GY28-6661, and IBM System/360 Operating System: MVT Supervisor, Program Logic Manual, GY28-6659). Other routines performing the master scheduling task establish the queue initialization task, and schedule the execution of the initial SET command and the commands specified by its AUTO parameter. When the system has been initialized, the Master Scheduler routines remain in the wait state except when an operator command or a system log operation requires the establishment of a system task. The Master Scheduler Attach routine then issues the ATTACH macro instruction to establish the START command task (which is executed by the System Task Control routine) in response to START commands, and to establish the appropriate command execution tasks in response to other task creating commands. When the required task has been established, the master scheduler returns to the wait state; when the tasks it establishes are terminated, the routines performing those tasks return control to the supervisor.

Queue Management

The queue data set, which contains the work queues, is initialized or reinitialized by routines performing the queue initialization task. The space for the queue data set is allocated when the system is generated; the data set is not, however, placed in the proper format until, in response to the F parameter of the initial SET command, the Queue Formatting routine writes initialized queue control records (QCRs) and sets up and chains logical tracks.

Once the queue data set has been formatted, the procedure need not be repeated; when the system is reloaded the F parameter may be omitted from the initial SET command. When the F parameter is omitted, System Restart routines, performing the queue initialization task, inspect the queue data set to insure that there are no incomplete input queue entries, and to re-enqueue any dequeued input or output queue entries so that they can be processed when an initiator and system output writer are started.

Management of the work queues is done by a set of routines that are used as subroutines by routines performing system tasks. There are 76 queues, of which 56 are used by the operating system. These are:

- The 15 input queues, which contain entries describing jobs to be run by the system.
- The Remote Job Entry (RJE) queue, which contains entries describing jobs for input to the job management routines.
- The hold queue, which contains input queue entries in the hold state as a result of a HOLD command.
- The 36 output queues, which contain entries describing the system output of jobs that have been run by the system.
- The free-track queue, which contains the tracks in the queue data set that are not assigned to a queue entry.
- The automatic SYSIN batching (ASB) queue, which contains compressed JCL statements for use as input to the interpreter.
- The background reader (BRDR) queue, which contains pointers to data sets for TSO jobs submitted via the SUBMIT command. (A description of the background reader is in the TSO Control Program PLM. A description of the

SUBMIT command is in the TSO Command Processor PLMs.)

The user of queue management provides a parameter area that specifies the operation to be performed; when records are to be transferred, he also provides a main storage area for the records. The user then links to a Queue Management routine, which performs one of the following functions:

- It initializes for the establishment of a queue entry.
- It assigns records and (as required) logical tracks to the entry.
- It writes records into the assigned locations, or reads them into main storage.
- It enqueues queue entries in the specified queue, at the specified priority.
- It dequeues the highest priority entry from the specified queue.
- It deletes entries: returns the tracks assigned to them to the free-track queue.

An option permits logical tracks to be "stacked" in main storage, thus reducing the number of accesses to the queue data set. The Track Stacking routines, which are subroutines of the Queue Management routines, establish and maintain the track stack in main storage and accomplish the transfer of logical tracks and records.

Operator Commands

Operator commands may be entered into the system via an operator's console input device, via a TSO terminal, or via a system input reader. The execution of operator commands is scheduled by an SVC routine (SVC 34); the commands are executed either by a routine performing a task established (by the master scheduler) as a command execution task, or by routines performing existing system tasks.

The command scheduling routines (SVC 34) examine the command verb and determine whether the command is to be executed immediately by a routine that is part of an existing task or whether execution of the command requires the creation of a new system task. If the command is to be executed immediately, control passes to the appropriate processing routine via an XCTL macro instruction. If the command requires the creation of a system task for execution, a command scheduling control block (CSCB) is built, the command is stored in the CSCB, and the master scheduling task is posted. The master scheduling task passes control to the appropriate command processing routine via an ATTACH macro instruction.

If the command to be executed is a START or MOUNT command, the master scheduler passes control to the System Task Control routine via the ATTACH macro instruction. The System Task Control routine obtains a region, checks the operand of the command, and builds an "internal input stream" from the parameters in the command operand. The interpreter, used as a closed subroutine, reads the input stream (which calls a cataloged procedure), and creates the appropriate job description tables. The System Task Control routine then passes control to the Initiator subroutine to complete the command processing.

The Initiator subroutine uses the I/O Device Allocation routine to allocate devices and then attaches the program that executes the starting task. After execution and termination of the starting task, the initiator returns control to the System Task Control routine. This routine frees the region and returns control to the master scheduler.

The queue accessing commands (CANCEL, DISPLAY, HOLD, RELEASE, and RESET) are executed by routines performing the queue alter task. These routines access the input, hold, ASB, RJE, BRDR, and output queues and make changes or display information as specified in the command.

The remaining commands are discussed in the "Command Processing" section of this publication, and in the sections describing the executing routines.

Part 1: Initialization and Restart

When the operating system is loaded, it must be initialized to conform to the locations and extents of the system data sets, and to the requirements of the installation. This process, which includes formatting the work queue data set, is called system initialization. If the work queue data set is already in the proper format, special processing must be performed to purge the work queues of incomplete and inappropriate entries; in this case, the processing is called system restart. System initialization and restart are performed by routines executing the master scheduling task, or by routines executing subtasks of the master scheduling task.

The Master Scheduling Task

The master scheduling task (Figure 1) is one of the system tasks established when the system is loaded. The portions of this task that are discussed in this publication are to perform the following functions:

- Initializing the master scheduler resident data area.
- Establishing the queue initializing task.
- Establishing the direct access volume initializing task.
- Causing the initializing commands to be executed.
- Initializing the system management facilities (SMF) if they are included in the system.
- Establishing command processing tasks (see the "Command Processing" section of this publication).

The remaining portions of the master scheduling task are discussed in other publications. The portion performed by the nucleus initialization program (NIP) is discussed in the IPL/NIP Program Logic Manual; the portions performed by the Console Initialization routine and by the Log Initialization routine are discussed in the MVT Supervisor program logic manual; and the portions performed by the TSO Initialization routine are discussed in the Time Sharing Option Control Program Program Logic Manual.

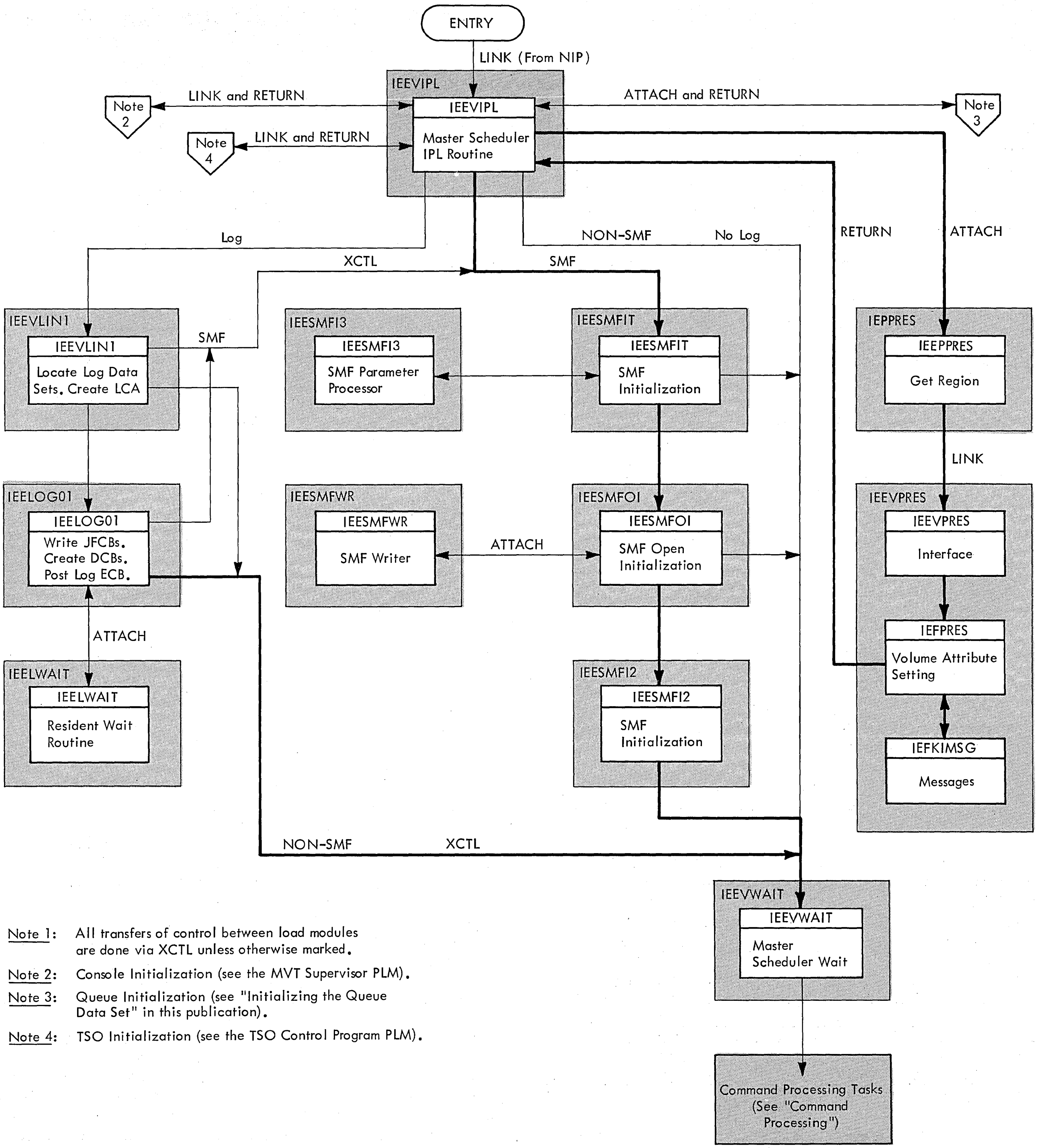
The initializing functions of the master scheduling task are performed in the Master Scheduler IPL routine (module IEEVIPL), which the macro SGIEE00V assembles during system generation. The Nucleus Initialization program enters the Master Scheduler IPL routine via a LINK macro instruction, and passes to it a pointer to the UCB for the parameter library (SYS1.PARMLIB) unit.

Before performing the initialization of the master scheduler resident data area, the Master Scheduler IPL routine links to the Console Initialization routine (IEECVCTI). On the return, it posts the event control block (ECB) that the communications task is waiting for. Next, the IPL routine issues a 'READY' message for the operator and moves the master scheduler's TIOT from the IPL routine into a section of main storage obtained from subpool 255. Then the routine uses a WTO macro instruction to display commands that are available for execution as a result of the AUTO parameter in a SET command, and waits for the SET command to be issued.

The SET command causes the Command Scheduling routine to issue a POST macro instruction that specifies an ECB in the master scheduler resident data area. When the posting occurs, the Master Scheduler IPL routine looks for the procedure library and for the work queue data sets. If the SET command has specified the units that contain those data sets, the routine checks the specified units; if no units are specified in the SET command, the routine checks the catalog, the units specified when the system was generated, and finally the system residence volume.

When the routine has determined which units should contain the data sets, it checks to see that the units are in the ready state, that the correct volumes are mounted, and that the data sets are actually on the specified volumes. If so, it stores pointers to the UCBs in the master scheduler resident data area, and in the master scheduler's TIOT. Then the IPL routine catalogs the procedure library data set into the volume on which it was found.

If either data set cannot be found, or if the correct volumes are not mounted, or if the specified units are not ready, the routine issues a message to the operator and waits for the SET command to be re-issued. Initialization does not continue until the error has been corrected and the data sets are found.



Note 1: All transfers of control between load modules are done via XCTL unless otherwise marked.

Note 2: Console Initialization (see the MVT Supervisor PLM).

Note 3: Queue Initialization (see "Initializing the Queue Data Set" in this publication).

Note 4: TSO Initialization (see the TSO Control Program PLM).

Figure 1. The Master Scheduling Task

In System/370 machines, the Master Scheduler IPL routine uses the WTOR macro instruction to ask the operator for SET command parameters for setting the time-of-day (TOD) clock. After the operator has replied, the routine generates an internal SET command and issues SVC 34 to pass control to the Command Scheduling routine. During command processing, the TOD Clock routines (modules IGC6503D and IGC6603D) of the Command Scheduling routine use the CLOCK and DATE parameters from the SET command to set the TOD clock (see the "Command Processing" section of this publication).

After cataloging the procedure library, the IPL routine uses the ATTACH macro instruction to pass control to the queue initializing task (in module IEFQINTZ) and then waits for completion of the initialization procedures. After initialization is completed, the IPL routine receives control and uses the DETACH macro instruction to detach the TCB of the queue initialization task. Then the routine issues the ATTACH macro instruction to establish the direct access volume initialization task (module IEEPPRES) and waits for the completion of the task.

When the ATTACH macro instruction is issued, control is passed to the Get Region routine (module IEEPPRES), which obtains a register save area and a region of main storage, then passes control (via a LINK macro instruction) to the Interface routine.

After obtaining a pointer to the UCB for the parameter library unit, the Interface routine (module IEEVPRES) opens a DCB for the parameter library, obtains main storage for a read DECB, and branches to the Volume Attribute Setting routine.

The Volume Attribute Setting routine (module IEFPRES) scans the PRESRES data set entries until it finds one that corresponds to a mounted volume, then sets the volume attributes in the UCB.

If there are volumes listed in the PRESRES data set that are not mounted, the routine uses the WTOR macro instruction to request the operator to mount the correct volumes. The operator replies with a list of unit addresses or "GO"; if the reply contains unit addresses, the routine reads the volume labels, checks the volume serials, and places them (and the volume attributes) in the UCBs.

When all of the volumes listed in the PRESRES data set are mounted, or when the operator has replied "GO", the routine uses the WTO macro instruction to inform the operator of the volume serials and attributes of all permanently resident and reserved volumes, then returns control to the master scheduler.

The Master Scheduler IPL routine then detaches the TCB for the direct access volume initializing task and uses the WTO macro instruction to display the commands selected for execution via the AUTO parameter of the SET command. It schedules the execution of the commands by issuing an SVC 34 for each command.

If the system includes TSO, the IPL routine uses a LINK macro instruction to give control to module IEEVSIPL for initializing the SYS1.BROADCAST data set. When the IPL routine again receives control, it uses the XCTL macro instruction to pass control either to the system log initialization routine (IEEVLIN1), if the system includes the system log, or to the SMF initialization routine (IEESMFIT), if the system includes the system management facility. If the system contains neither the SMF nor the system log facility, the routine uses the XCTL macro instruction to give control to the Master Scheduler Wait routine (IEEVWAIT).

Routines IEEVLIN1 and IEELOG01 initialize the system log. Routine IEEVLIN1 locates the log data sets and establishes the Log Control Area (LCA) and log buffers. Routine IEELOG01 writes the log job file control blocks onto the job queue, creates the log DCBs, issues an ATTACH macro instruction for the log Writer routine (IEELWAIT) and posts the log ECB.

If module IEEVLIN1 does not locate the log data sets, the operator receives a message that the system does not support the log option. As a result, module IEE0303F (which is one of the SVC 36, or WTL, routines) re-issues WTL macro instructions as WTO macro instructions. The control program (module IEE1603D) then treats LOG and WRITELOG commands (from the operator) as NOPs and sends a message to the console informing the operator that it does not support the system log. The log initialization routine IEELOG01 then uses an XCTL macro instruction to pass control either to the SMF initialization routine if the system includes SMF or to the Master Scheduler Wait routine IEEVWAIT.

The posting of the log ECB (by routine IEELOG01) causes the log writer routine IEELWAIT to become dispatchable (executable) to continue the system log initialization. When routine IEELWAIT is given control, it determines that initialization processes are being performed and uses the WTL macro instruction to open the log data sets. When the log data sets are successfully opened, routine IEELWAIT waits on the log ECB. (Section 7 of the MVT Supervisor PLM contains further information about the system log.)

Initializing the system management facilities includes obtaining and storing the SMF parameters, allocating devices to and opening the SMF data sets, establishing the SMF task, initializing a 10-minute timer, and issuing the initial SMF records. These functions are performed by the SMF Initialization routine, the SMF Open Initializer, and the SMF SVC routine (SVC 83).

The SMF Initialization routine consists of three modules: IEESMFIT, IEESMFI3, and IEESMFI2. The routine is entered at IEESMFIT, which adds the compiled-in DD names SMFMANX and SMFMANY to the master scheduler TIOT, then obtains main storage from subpool 255 for the system management control area (SMCA). The SMCA is described in Appendix A.

Module IEESMFIT of the Initialization routine stores a pointer to the SMCA in the communications vector table (CVT) then determines whether the SYS1.PARMLIB data set contains the SMF parameter member, SMFDEFLT. If not, module IEESMFIT passes control to module IEESMFI3 at entry point IEESMFMS. If the SMF parameter member is present, module IEESMFIT opens the data set, and reads member SMFDEFLT into main storage. If an I/O error occurs during the read, the routine passes control to IEESMFI3 at entry point IEESMFIO; otherwise, it passes control to IEESMFI3 at entry point IEESMFI3.

Module IEESMFI3 inspects the SMF parameter member to determine whether all required parameters have been supplied correctly. If any required parameters are missing or incorrectly specified, or if the routine was entered because the member was missing or there was an I/O error, IEESMFI3 uses the WTOR macro instruction to have the operator enter the parameters.

When the parameters have been entered correctly, the routine stores them in the SMCA and uses the WTO macro instruction to display them to the operator. If operator intervention is permitted (OPI=YES), the routine uses the WTOR macro instruction to allow the operator to make changes. When it has stored any changes, IEESMFI3 passes control to IEESMFIT at entry point IEESMFI4.

When it is entered from IEESMFI3, module IEESMFIT determines whether an SMF data set was specified. If not, the routine uses the XCTL macro instruction to pass control to the Master Scheduler Wait routine (module IEEVWAIT); if so, the routine obtains main storage from subpool 255 for the SMF buffer, and uses the XCTL macro instruction to pass control to the SMF Open Initializer routine.

The SMF Open Initializer prepares to have the SMF data sets allocated and opened, and establishes the SMF task. The JFCBs for the data sets (SYS1.MANX and SYS1.MANY) are compiled into module IEESMFOI; the routine uses the Queue Management Assign/Start and Read/Write routines to obtain space in the work queue data set and to write the JFCBs[1]. It stores the JFCB addresses in the master scheduler TIOT, and uses the ATTACH macro instruction to pass control to the SMF Writer routine (module IEESMFWR). The SMF Writer routine issues a WAIT macro instruction (specifying the writer ECB in the SMCA) to place the SMF task in the wait state.

The SMF Open Initializer then sets the first time switch in the SMCA, posts the writer ECB causing the SMF Writer routine to issue SVC 83 with a negative pointer to the SMCA for allocation and opening of SMF data sets, and issues a WAIT macro instruction for the buffer ECB.

The SMF SVC routine (SVC 83) consists of three load modules: the Record Transfer routine (module IEESMF8C), the SMF Open routine (module IEESMFOP) and the SMF Allocation routine (module IEESMFAL). The Record Transfer routine, which is always entered first, inspects register 1 to determine whether a record transfer is required. When the SVC is to perform data set switching or initialization functions, the register is negative, and the Record Transfer routine uses the XCTL macro instruction to pass control to the SMF Open routine.

The SMF Open routine (module IEESMFOP) determines whether to perform initialization or data set switching by testing the first-time switch (bit 3 of the SMCA miscellaneous indicators field). If the switch is on, initialization is required; the routine loads register 5 with a pointer to the first data set area in the SMCA and inspects the corresponding entry in the master scheduler TIOT. If the TIOT UCB pointer field is zero, the SMF Open

----

[1] The JFCB records appear as an incomplete input queue entry. In the event of a system restart, the System Restart routines return the space occupied by the JFCB records to the free-track queue, and the SMF Open Initializer routine replaces them when it is executed again.

routine uses the XCTL macro instruction to pass control to the SMF Allocation routine.

The SMF Allocation routine (module IEESMFAL) uses register 5 to determine which of the two SMCA data set areas contains information on the data set to be allocated. The user may specify either a volume serial or a unit address for each SMF data set, and the SMF Allocation routine uses the specified information as a search argument in a search of the UCB list. When it finds a match, the routine stores a pointer to the UCB in register 1.

If the device is a tape device, or if it is a direct access device capable of handling the maximum size SMF record, the SMF Allocation routine marks the UCB "allocated" and "permanently resident." If the device is a direct access device, the routine also turns on bit 2 in the appropriate device status field in the SMCA.

When it has completed the allocation of a device to one SMF data set, the SMF Allocation routine sets the SMCA Open DS routine switch to X'80', then uses the XCTL macro instruction to pass control to the SMF Open routine.

When the SMF Open routine (module IEESMFOP) is entered, it tests the Open DS routine switch. Since it was entered from the SMF Allocation routine, the switch is set to X'80' and register 1 contains a pointer to the UCB corresponding to the allocated device.

The Open routine sets the Open DS routine switch to zero, then stores the contents of register 1 in the TIOT entry corresponding to the data set associated with the allocated device. The routine uses the RDJFCB macro instruction to bring the JFCB for the data set into main storage, then updates the JFCB volume serial and label type fields with information from the UCB and SMCA. Next, the routine uses the OBTAIN macro instruction to determine whether the data set is empty. If so, the routine uses the OPEN macro instruction to open the data set for BSAM.

When the data set has been opened, the SMF Open routine tests the first-time switch. If the bit is on (in this case it will be, because this is the first post-allocation entry) the routine sets it off, then determines whether the data set is on a tape volume or a direct access volume.

If the data set is on a tape volume, no further allocation is required, because only one tape SMF data set is permitted. If, however, the data set is on a direct access volume, two data sets are required, and another device must be allocated. The Open routine sets register 5 to point to the other data set area in the SMCA and uses the XCTL macro instruction to pass control to the SMF Allocation routine.

When allocation is complete, the SMF Open routine is again entered with the Open DS routine switch set to X'80'. The routine stores the UCB pointer and opens the data set, then tests the first-time switch. In this case, the switch is off; no further processing is required, and the SMF Open routine issues the EXIT macro instruction to return control to the SMF Writer routine, via the supervisor.

On the return from the SVC 83 routine, the SMF Writer routine posts the buffer ECB that module IEESMFOI issued the WAIT on. The Writer routine then issues a WAIT on the writer ECB in the SMCA. The SMF Open Initializer routine determines whether allocation and opening of the SMF data sets was successful. If either failed, the routine passes control via an XCTL to the Master Scheduler Wait routine (module IEEVWAIT). If allocation and opening were successful, the routine issues an XCTL to pass control to module IEESMFI2.

Module IEESMFI2 branches to the Timer Enqueue routine (see the MVT Supervisor PLM) and passes it a compiled-in timer queue element that requests 10-minute time intervals.

If an SMF data set is present, module IEESMFI2 constructs the SMF IPL record (type 0), and the SMF online devices record (type 8), and if volume information is requested, the SMF online direct access devices record (type 19). It uses SVC 83 to have the records transferred to the SMF buffer, then uses the XCTL macro instruction to pass control to the Master Scheduler Wait routine.

The Master Scheduler Wait routine (module IEEVWAIT) enters the wait state, and the system is ready to process the commands that start the job management process.

Initializing the Queue Data Set

The 76 work queues occupy space on a permanently resident volume. The space for the queue data set, which contains these queues, is allocated when the system is generated, in response to a DD statement submitted by the installation. The DD statement specifies the amount of space (contiguous tracks) to be allocated.

When the queue data set is initialized, it is divided into two major areas. One, the control record area, is fixed in length; it occupies the first 2736 bytes of the queue data set, and contains a 36-byte queue control record (QCR) for each of the 76 queues. The first QCR is the master QCR; it contains information about the queue data set as a whole, and is used in the control and maintenance of the free-track queue. The other QCRs are each used in the maintenance of a single queue; there is one hold queue control record, one ASB queue control record, one RJE queue control record, 15 input queue control records, 36 output queue control records, one background reader queue, and a queue control record for each of the 20 unused queues.

The other major area in the queue data set is the logical track area. A logical track is an area of contiguous space (not necessarily corresponding to a physical track) that contains a 20-byte logical track header (LTH) record, and a number, specified by the installation, of 176-byte records. The length of a logical track is 20+176N bytes, where N is the installation-specified number (between 10 and 255) of records per logical track.

The queue initializing process is begun when the Master Scheduler IPL routine attaches the initializing task, passing control to module IEFQINTZ. This module, which resides in the link-pack area, obtains a region, then passes control to module IEFSD055. This module obtains main storage for a work area, and for a DEB and DCB for the queue data set, then opens the data set for BSAM. The routine also examines the UCB for the jobqueue to determine if the queue is on a rotational position sensing (RPS) device.[1] If it is, the routine sets an indicator in the queue manager resident storage and stores the device type code. All other routines use this information prior to reading from or writing to the jobqueue.

FORMATTING THE QUEUE DATA SET

If the operator used the parameter "F" in the SET command, the queue must be formatted, and module IEFSD055 uses the WTOR macro instruction to give the operator the opportunity to override the queue parameters specified at system generation. These parameters (the number of 176-byte records per logical track, the number of 176-byte records to be reserved for an initiator, and the number of 176-byte records to be reserved for terminating a job) are discussed in the section "Assigning Queue Space." The parameters are stored in the work area, and control is passed to module IEFORMAT.

Module IEFORMAT uses BSAM to write records into the queue data set to put it in the proper format (the format of the queue data set after initialization is shown in Figure 2). The routine writes, into the control record area, a complete set of queue control records, all of which are set to zero. It then writes the logical track header records and intervening 176-byte data records into the logical track area. The 176-byte data records are set to zero; each LTH contains a pointer to the next LTH.

The pointer (designated NN) in the LTH is a 2-byte binary number that represents an offset from the beginning of the logical track area. The NN of the first LTH is one; the NNs of subsequent LTHs are found by adding the number of records per logical track, plus one, to the current NN.

Finally, module IEFORMAT builds the master queue control record with pointers (in NN format) to the first and last logical track headers. After control is returned to module IEFSD055, XDAP is used to write the master QCR (see Figure 3) into the control record area.

Module IEFSD055 moves the DEB, DCB and master QCR into the queue management resident data area. The three queue parameters T, K, and T' (which are described in the section "Assigning Queue Space") are set up; a bit is set to indicate that a reserve of tracks exists, and the routine closes the DCB before returning control to module IEFQINTZ. Module IEFQINTZ frees the region, and returns control to the Master Scheduler IPL routine.

----

[1] Rotational Position Sensing is an IBM 3330 Disk Pack characteristic that permits the location of records by an explicit angular position. Further details may be found in the Component Summary publication listed in the Preface.

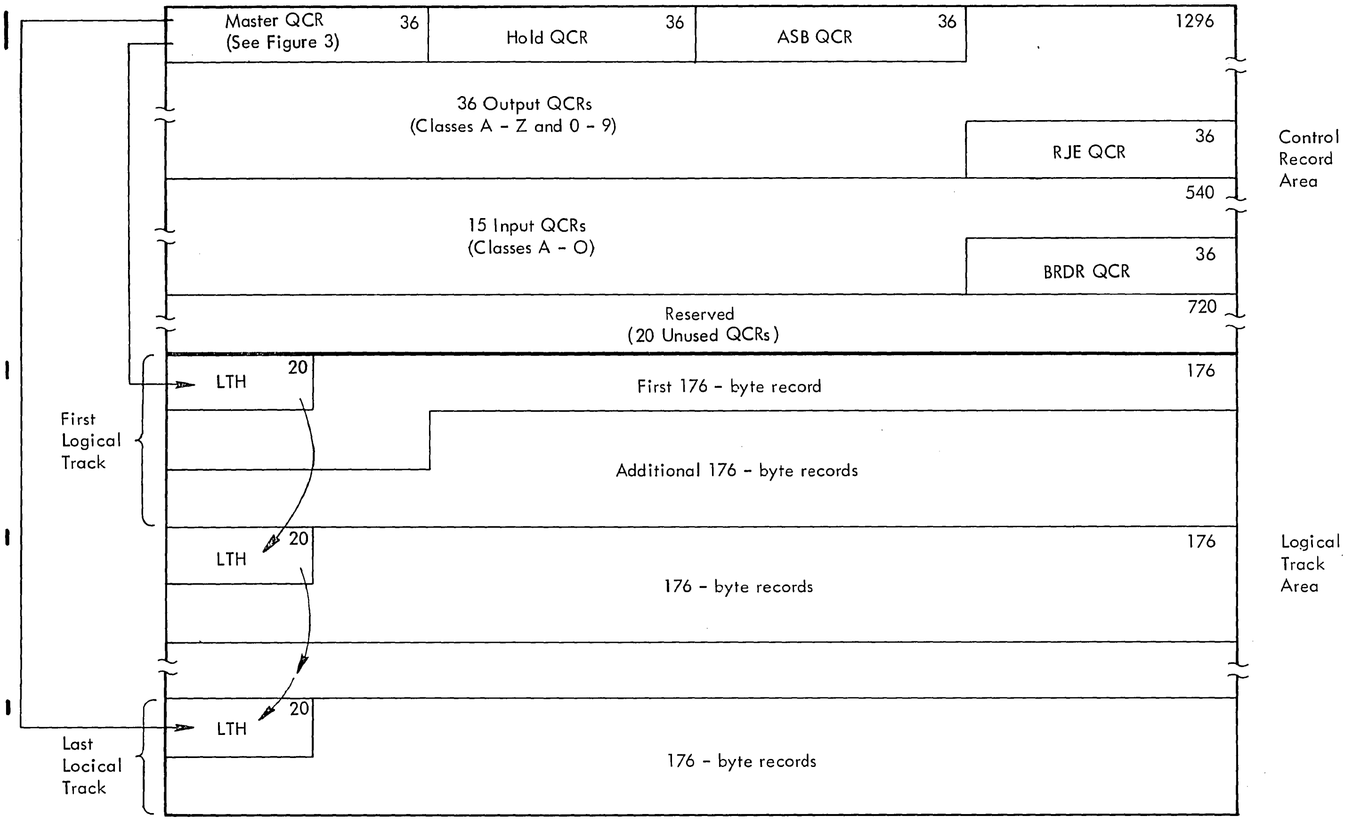


Figure 2. The Queue Data Set After Initialization

Address (MBBCCHHR) of the master QCR 8		
Reserved 1	Pointer to the First Logical Track in the Free-Track Queue 2	Reserved 1
Number of Logical Tracks in the Queue Data Set 2	Number of Logical Tracks in the Free-Track Queue 2	
Number of Logical Tracks Reserved for Job Termination 2	Number of Logical Tracks Reserved for Initiators 2	
Pointer to the Last Logical Track in the Free-Track Queue 2	Address (TT) of the First Physical Track Containing no QCRs 2	
Number of QCRs per Physical Track 2	Number of Records per Physical Track 2	
Number of 176-byte Records per Logical Track 2	Number of Logical Tracks to Reserve per Initiator 2	
Number of QCRs on the Mixed Track 2	Pointer to the First Record on the First Track Containing no QCRs 2	

Figure 3. Master Queue Control Record

SYSTEM RESTART

When the system is first initialized, the queue data set must be formatted; when the system is restarted, the formatting need not be repeated, but the existing queue data set must be reinitialized.

If the restart is necessary because of a power failure or system error, there may be entries in the queues for which processing had not been completed. The System Restart routines (shown in Figure 4) must therefore inspect the queue data set, determine the status of any queue entries that exist, and perform the processing required to allow normal processing to be resumed.

The following kinds of entries may be found in the queue data set. They are processed as described below:

- Incomplete input queue entries will exist if processing was halted while input streams were being processed. Such entries must be purged from the queue data set, and the logical tracks assigned to them returned to the free-track queue.
- Incomplete output queue entries may exist, reflecting the system output requirements of any jobs that have not been completely processed. If the input queue entry for a job is not complete (has not been enqueued) the logical tracks assigned to its output queue entries need only be returned to the free-track queue. Otherwise, the existing system output (and any system output generated when the job is processed) will be processed normally. Such entries are therefore identified, but not processed, by System Restart routines.
- Incomplete Automatic SYSIN Batching (ASB) queue entries will exist if processing was halted during execution of an ASB task. Logical tracks assigned to such entries will be returned to the free-track queue.

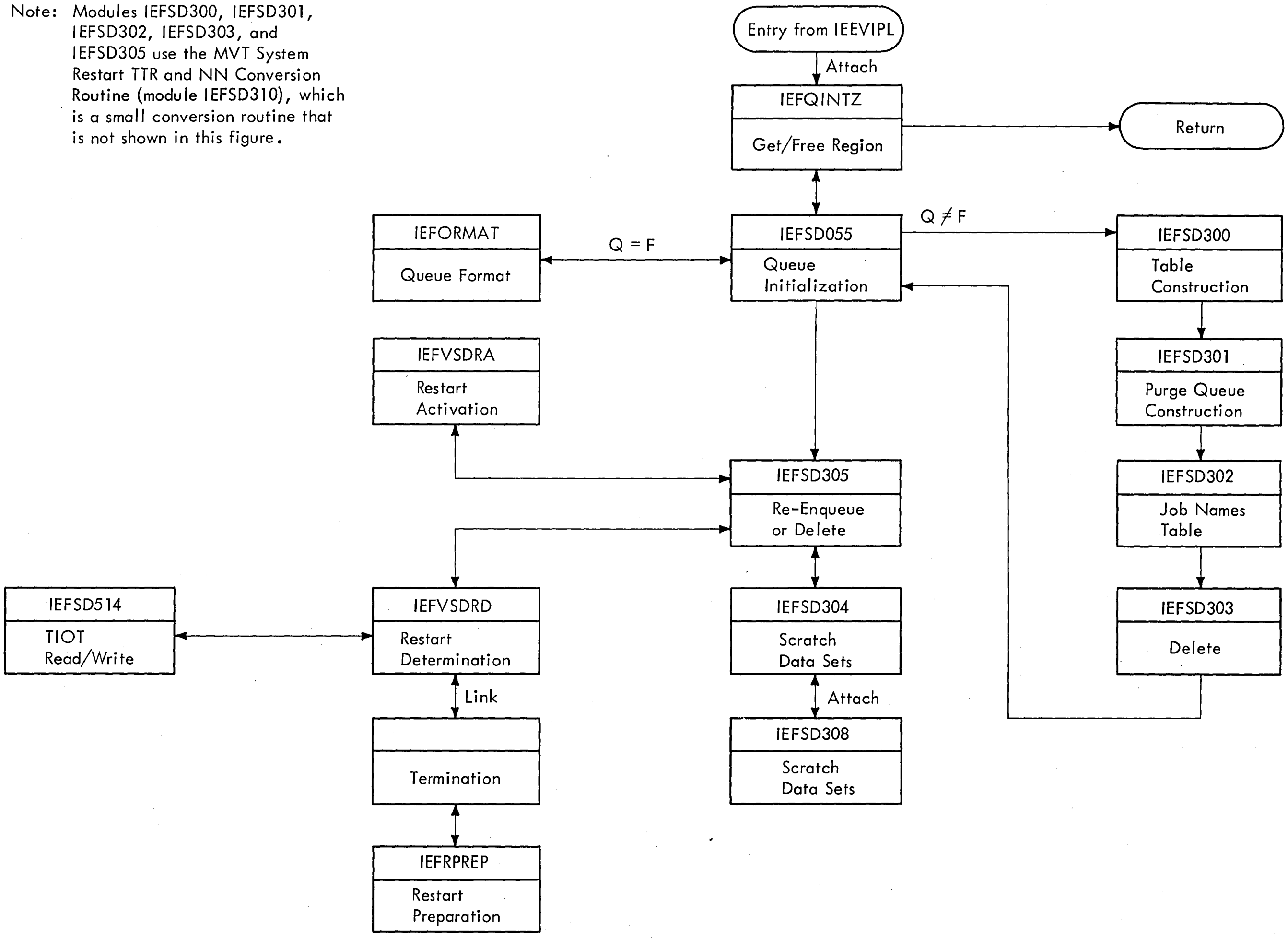


Figure 4. MVT System Restart

- Incomplete remote job entry (RJE) queue entries will exist if processing was halted during transmission to the central system. Logical tracks assigned to such entries will be returned to the free-track queue.

- Enqueued input, output, hold, ASB, and RJE queue entries may exist; they remain in the queues, and are processed normally when the appropriate system tasks are established.

- Dequeued input queue entries represent jobs that have been selected for initiation, but that have not been processed by the Job Termination routine. If such a job can be restarted, the System Restart routines initiate the restart; if the job cannot be restarted, the System Restart routines enqueue the job's output queue entries and delete its input queue entry.

- Dequeued output queue entries represent partially finished system output "jobs". Such queue entries are modified so that all system messages and data sets that have not been processed will be written by the first eligible output writer started.

- Dequeued hold queue entries may exist if the Queue Alter routine was processing an entry in the hold queue, as during execution of CANCEL, RESET and RELEASE commands. The System Restart routines re-enqueue such entries in the hold queue.

- Dequeued ASB queue entries represent jobs that have been processed by the ASB Input Stream Processor routine. Such entries are re-enqueued in the ASB queue, and any input or output queue entries for the job are deleted.

- Dequeued RJE queue entries represent jobs that are being processed by the RJE reader, or that have been processed by the RJE reader. Such entries are re-enqueued on the RJE queue. Any input or output queue entries associated with them are treated normally.

Table Construction

The operator indicates that the system is being restarted (that the queue data set is in the proper format) by omitting the "F" parameter from the SET command. When module IEFSD055 has opened the queue data set (see "Initializing the Queue Data Set"), it passes control to the Table Construction routine, which begins the restart process.

The Table Construction routine (module IEFSD300) obtains main storage for the system restart work area (Figure 5), reads the master QCR into main storage, and obtains main storage for the logical track header table (table A), the top-of-queue-pointer table (table B), and the queue entry pointer table (table C).

Table A (see Figure 6) is constructed with an entry for each logical track in the queue data set. As tracks are released to the free-track queue, and as queue entries are processed, the System Restart routines set the corresponding table A entries to zero.

Table B is simply a list of pointers, and is therefore not shown. The table enables the System Restart routines to find the first entry in each queue without having to re-read the QCR.

Table C (also shown in Figure 6) is constructed as queue entries are processed. It is used to find the SCD in entries that have corresponding system output queue entries, and enables the System Restart routines to read the SCDs in order by direct access storage address.

The Table Construction routine builds table B by reading the QCRs into main storage and extracting the top-of-queue pointer from each QCR. It constructs table A by reading each LTH into main storage (in their physical sequence) and extracting the required information from it, then converts each 2-byte pointer (in NN format) to a relative address within table A, using the formula

$$8\left[\left(\frac{NN}{X+1}\right)+1\right]$$

= relative address in table A of the LTH where X is the number of records per logical track.

When tables A and B have been constructed, the routine identifies, and sets to zero, all table A entries corresponding to tracks in the free-track queue. The master QCR furnishes a pointer to the first track; when the corresponding table A entry is found, its pointer to the next logical track assigned is extracted, and the entry is set to zero. If the pointer is zero, the entry corresponds to the last track in the free-track queue.

Table A entries corresponding to tracks assigned to enqueued input queue entries, hold queue entries, output queue entries, ASB queue entries and RJE queue entries are set to zero in a similar manner. Table B furnishes the starting point. When the input, hold, ASB, and RJE queues are processed, an additional step is performed;

the first-logical-track-assigned pointer for each entry is converted to a relative address in table C, and stored at that location.

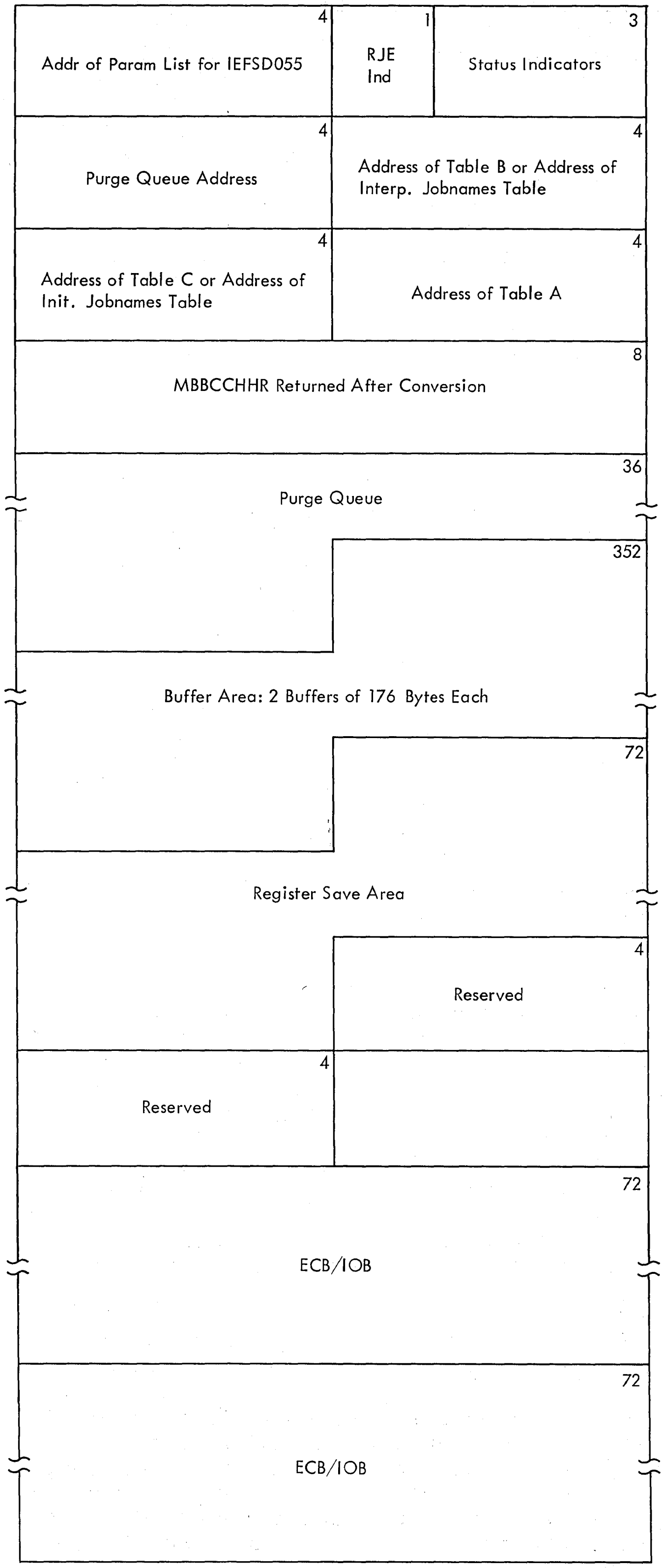

4
Addr of Param List for IEFSD055
1
RJE Ind
3
Status Indicators
4
Purge Queue Address
4
Address of Table B or Address of Interp. Jobnames Table
4
Address of Table C or Address of Init. Jobnames Table
4
Address of Table A
8
MBBCCHHR Returned After Conversion
36
Purge Queue
352
Buffer Area: 2 Buffers of 176 Bytes Each
72
Register Save Area
4
Reserved
4
Reserved
72
ECB/IOB
72
ECB/IOB

Figure 5. System Restart Work Area

When this processing has been completed, table A contains entries corresponding to the following kinds of queue entries:

- Incomplete input queue entries.
- Incomplete ASB queue entries.
- Incomplete RJE queue entries.
- Incomplete output queue entries.
- Dequeued input queue entries.
- Dequeued hold queue entries.
- Dequeued ASB queue entries.
- Dequeued RJE queue entries.
- Dequeued output queue entries.

In addition, the construction of table C has been started; it contains an entry corresponding to the first logical track assigned to each enqueued input, hold, ASB, and RJE queue entry.

Purge Queue Construction

The storage occupied by table B is then released, and control is passed to the Purge Queue Construction routine (module IEFSD301), which obtains main storage for the jobnames table (see Figure 6), then scans table A to identify the remaining queue entries by inspecting the queue and status fields of each table A entry.

Incomplete Input Queue Entries

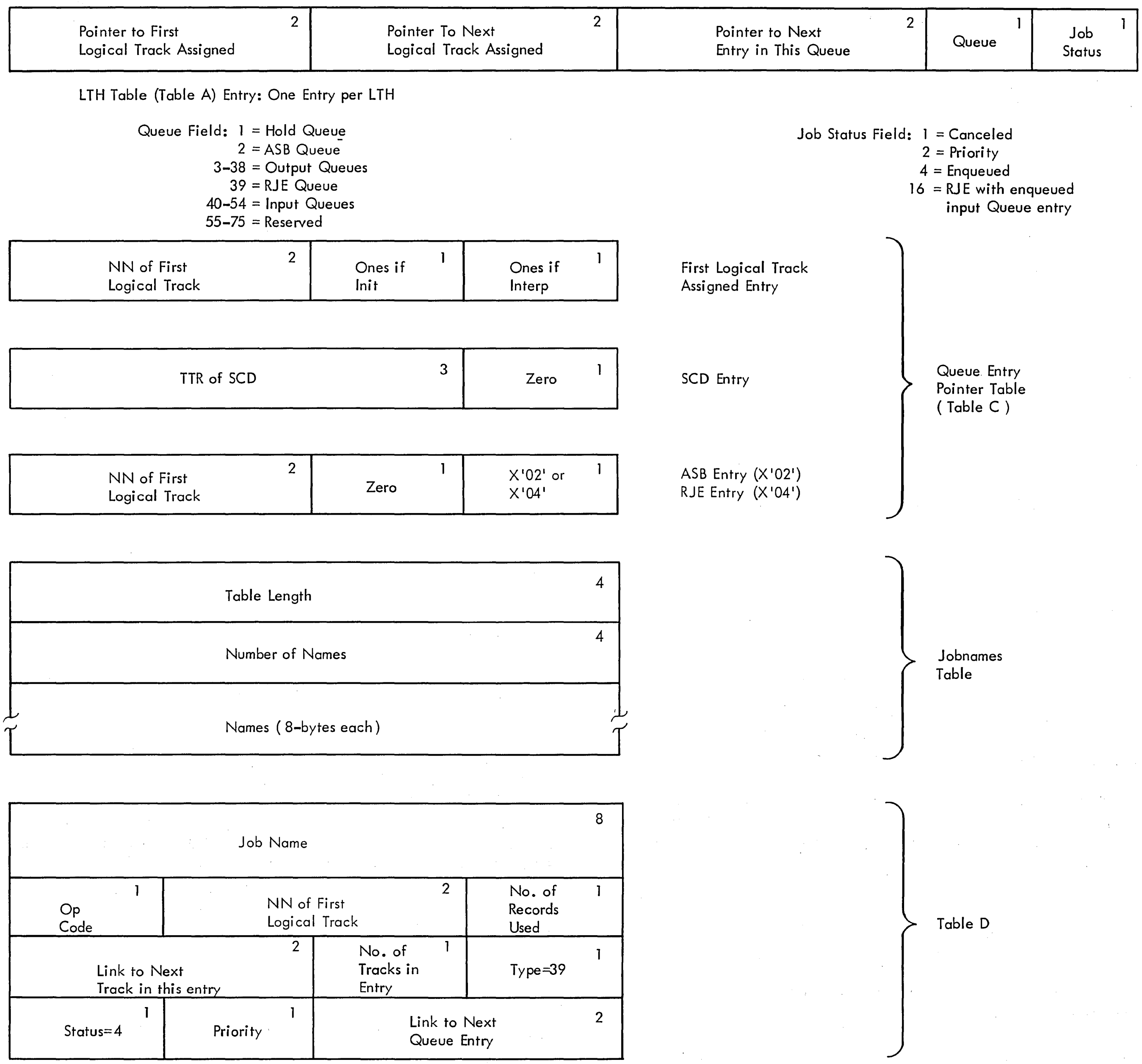
The relative address in table A of the first logical track assigned to the queue entry is divided by two, which converts it from a relative address in table A to a relative address in table C. The resulting address is stored in table C, and the table A entries corresponding to the tracks assigned to the queue entry are marked "tested."

Incomplete Output Queue Entries

No corresponding entry is made in table C, but the table A entries are marked "tested."

Incomplete ASB and RJE Queue Entries

The relative address in table A of the first logical track assigned to the queue entry is divided by two, which converts it from a relative address in table A to a relative address in table C. The resulting address is stored in table C, and the table A entries corresponding to the tracks assigned to the queue entry are marked "tested."



Pointer to First Logical Track Assigned	Pointer To Next Logical Track Assigned	Pointer to Next Entry in This Queue	Queue	Job Status
2	2	2	1	1

LTH Table (Table A) Entry: One Entry per LTH

Queue Field:	
1	= Hold Queue
2	= ASB Queue
3-38	= Output Queues
39	= RJE Queue
40-54	= Input Queues
55-75	= Reserved

Job Status Field:	
1	= Canceled
2	= Priority
4	= Enqueued
16	= RJE with enqueued input Queue entry

Queue Entry Pointer Table (Table C)

First Logical Track Assigned Entry

NN of First Logical Track	Ones if Init	Ones if Interp
2	1	1

SCD Entry

TTR of SCD	Zero
3	1

ASB Entry (X'02')
RJE Entry (X'04')

NN of First Logical Track	Zero	X'02' or X'04'
2	1	1

Jobnames Table

Field	Length
Table Length	4
Number of Names	4
Names (8-bytes each)	

Table D

Job Name 8			
Op Code 1	NN of First Logical Track 2		No. of Records Used 1
Link to Next Track in this entry 2		No. of Tracks in Entry 1	Type=39 1
Status=4 1	Priority 1	Link to Next Queue Entry 2	

Figure 6. System Restart Tables

Dequeued Input Queue Entries

These entries are enqueued in the purge queue, with priority 14. The purge QCR is maintained in main storage, and the enqueue function is performed in module IEFSD301. When the entry has been enqueued, the relative address in table A of the first logical track assigned to the entry is divided by two to convert it to a relative address in table C, and the result is stored in table C. All table A entries corresponding to logical tracks assigned to the queue entry are set to zero.

Dequeued Hold Queue Entries

The procedure used is the same as that used for dequeued input queue entries.

Dequeued ASB and RJE Queue Entries

The relative address of the entry's first logical track header record is placed in Table C; a hexadecimal 02 (ASB) or 04 (RJE) is placed in the status field of the Table C entry, and the entry is enqueued in the purge queue at priority 14.

Dequeued Output Queue Entries

The procedure used is the same as that used for dequeued input queue entries, except that no table C entry is made.

When processing of table A has been completed, the pointers in table C are converted to NN format, with the following formula:

$$\left[\frac{\text{Pointers from Table C} - 1}{4}\right](X+1) + 1 = NN$$

where X is the number of records per logical track.

All table A entries have been set to zero, except those entries that correspond to the logical tracks assigned to incomplete input, output, ASB, and RJE queue entries. Table C now contains a pointer (in NN format) to each incomplete, enqueued, and dequeued input queue entry, to each incomplete and dequeued ASB and RJE queue entry, and to each enqueued and dequeued hold queue entry. The purge queue now contains all dequeued input, output, hold, ASB, and RJE queue entries, enqueued with priority 14.

Control is now passed to module IEFSD302, which builds the jobnames table, and updates SCDs as required. The routine scans table C, and processes the non-zero entries as follows.

Incomplete Input Queue Entires

The pointer from table C is converted to MBBCCHHR format and the LTH is read into main storage. The job name is extracted from the LTH and placed in the first available slot in the interpreter jobnames table.

Enqueued and Dequeued Input Queue Entries

The pointer from table C is increased by 1 (to obtain the NN of the JCT) and is converted to MBBCCHHR format. The JCT is read into main storage. The address (in TTR format) of the SCD is extracted and placed in table C, replacing the pointer to the first logical track assigned to the queue entry.

Dequeued Input Queue Entries

If the table C entry represents an entry that has been dequeued from the input queue, an additional step is performed: the job name is extracted from the JCT and placed in the first available location in the initiator jobnames table.

Dequeued RJE Queue Entries

The entry's logical track header is placed in table D. When jobnames table construction is complete, the jobname of the RJE entry is compared to the interpreter names in the jobnames table; if a match is found, the name is removed from the jobnames table, and a switch is set to indicate that the input queue entry associated with the RJE queue entry has not been enqueued.

SCD Processing

Module IEFSD302 sorts the SCD pointers in table C (which are in TTR form) into ascending order, then uses them to read each SCD into main storage. An SCD is created by an interpreter and included in each input queue entry; it points to the output queue entries required for the job, and contains information from the QMPA used to build the queue entries. When an interpreter enqueues an input queue entry, all records required for the output queue entries have been assigned to them (except for the message class queue entry, which will require space for initiator messages); the SCD correctly reflects the status of the output queue entries while the entry is in the input queue. Therefore, if the SCD read into main storage was obtained from an enqueued input queue entry, the SCD needs no further processing. It is, however, used as a source of information; its pointers to the first logical track assigned to the output queue entries for the job are used to find the corresponding table A entries, which are then set to zero.

If, however, the entry from which the SCD was obtained has been dequeued, the SCD may not reflect the status of the output queue entry for the message class. When an input queue entry is dequeued by an initiator, information is extracted from the message class slot of the SCD and used to build a QMPA, and to assign additional records to the message class entry as they are needed. The SCD, however, is not updated to reflect the additional assignments, and therefore it contains obsolete information about the status of the message class entry.

In addition, since the number of records assigned (within a logical track) is not written into the LTH until the next logical track is assigned, the LTH of the last logical track assigned may also contain obsolete information. The number of records actually assigned in a logical track can be determined only from the QMPA used to build the entry, and when the system is being restarted, that QMPA is not available.

When the SCD is obtained from a dequeued input queue entry, it must be updated to accurately reflect the status of the output queue entry for the message class. The table A entries corresponding to the tracks assigned to the output queue entries are scanned; if they are zero, it means that the output queue entries have been enqueued, and the SCD slots corresponding to any enqueued output queue entries are set to zero.

If the table A entries corresponding to the logical tracks assigned to the message class output queue entry are not zero, the LTHs and the message class slot in the SCD must be updated. The LTH (in table A) of the last logical track known to be assigned is scanned. If the pointer to the next logical track is not zero, each succeeding LTH is scanned, until the LTH of the last logical track actually assigned has been found. The SCD is updated to reflect the actual status. The table A entries corresponding to the output queue entries are set to zero, and the SCD is written back into the queue.

When all SCDs have been read in and processed, all table A entries have been set to zero, except for those entries corresponding to the tracks assigned to incomplete input queue entries, and to the incomplete output queue entries associated with those jobs. The main storage occupied by table C is then released, and control is passed to module IEFSD303.

Returning Logical Tracks

Module IEFSD303 is then used to add those logical tracks assigned to incomplete queue entries to the free-track queue. Table A is scanned, and an LTH is constructed for each non-zero entry. The LTHs are chained together and written into the queue data set; the master QCR (passed by module IEFSD055) is updated to reflect the additional tracks and written to the queue data set, the main storage occupied by table A is released, and control is returned to module IEFSD055.

Module IEFSD055 moves the DEB, DCB, and master QCR into the master scheduler resident data area, then closes the DCB. In the restart situation, it then passes control to module IEFSD305.

Purge Queue Processing

Module IEFSD305 uses the WTO macro instruction to inform the operator of the names of any jobs represented by incomplete input queue entries. It then scans the purge QCR, updates the corresponding queue entries as necessary, and re-enqueues the entries in the appropriate queue. There are four kinds of entries in the purge queue; they are processed as described below.

Dequeued Output Queue Entries

The routine reads each 176-byte record into main storage. If the record is a system message block (SMB), it is ignored, and the next record is read in. If the record is a data set block (DSB), an OBTAIN macro instruction is used to determine whether the data set exists. If it exists, the next record is read from the queue entry; if not, the DSB is set to zero (except for the link field) and written back into the queue data set before the next record is read. Finally, the entry is reenqueued (by the Queue Management Enqueue routine) in the appropriate output queue, with a priority of 14.

Dequeued Input Queue Entries

If the entry is a dequeued input queue entry, the System Restart routines determine the point at which processing stopped, and process the entry accordingly:

- If processing stopped after a step of the job was selected, but before it started execution, the System Restart routines enqueue the job's system output queue entries and delete the job's input queue entry.

- If processing stopped while a step of the job was being executed, the System Restart routines determine whether the job is to be restarted: If so, the restart is initiated; if not, the routines enqueue the job's system output queue entries and delete its input queue entry.

- If processing stopped while a step (other than the last step) of the job was being terminated, the System Restart routines complete the termination and re-enqueue the job's input queue entry.

- If processing stopped while the last step of the job was being terminated, the System Restart routines complete the termination and delete the job's input queue entry.

The Restart Determination routine (module IEFVSDRD) determines the point at which the processing of the step stopped, and either returns control to module IFFSD305 or sets up an interface with the Termination routine and then passes control to it.

If processing stopped after the step was selected but before it started executing, the Restart Determination routine adjusts the SMB pointer in the SCT so that it points to the last Interpreter SMB for the step and returns control to module IEFSD305 with a return code X'10'.

If processing stopped while the step was being executed or terminated, the Restart Determination routine sets up an interface with the by Termination routine by:

- Testing bit 2 of the SCT step type indicators field; if the bit is set to one it indicates that there is system output data in the message class, and the Restart Determination routine updates the SMB pointer in the SCT to point to the first Termination SMB. If, however, the message class contains only operating system messages, the routine updates the pointer so that it points to the last Allocation SMB.

- Using the Table Breakup routine (module IEFSD514) to read the job step TIOT from the queue data set (where it was stored by the initiator). If an I/O error prevents the routine from reading the TIOT, control returns to module IEFSD305 with a return code of X'0C'.

- Constructing and initializing an LCT and setting the system restart bit in the JCT restart switches field (if the TIOT was read properly).

- Constructing a TCB for the job step. If processing stopped while the step was being terminated, the JCT points to the SCT for the next step; if processing stopped while the step was being executed, the JCT points to the SCT for the current step, and the Restart Determination routine sets the TCB ABEND flags on and sets the TCB completion code field to X'FF3'.

When it has set up the interface, the Restart Determination routine uses the LINK macro instruction to pass control to the Termination routine. The Termination routine performs its normal processing (except that it does not unallocate I/O devices), then returns control to the Restart Determination routine.

On the return, the Restart Determination routine (module IEFVSDRD) tests the return code passed to it by the Termination routine. If the return code is X'08', it means that processing stopped while the step was being executed. The Termination routine used the Restart Preparation routine and determined that a restart is possible, was requested, and has been authorized by the operator. In this case, the Restart Determination routine passes control (and the return code), to module IEFSD305.

If the return code passed by the Termination routine is X'00'. it means one of two things:

- Processing stopped while the step was being executed, but the step is not to be restarted.

- Processing stopped while the step (which is not the last step of the job) was being terminated; entry to the Termination routine from the restart Determination routine was "normal"; the Restart Preparation routine was not executed, and step termination was completed.

When it finds a return code of X'00', the Restart Determination routine tests the job failed bit in the JCT to determine whether the job was being processed in flush mode. If not, the bit is off, and the Restart Determination routine passes a return code of X'08' to module IEFSD305.

If the job was being processed in flush mode when processing stopped, the job failed bit is on, and the Restart Determination routine passes the X'00' return code to module IEFSD305.

If the return code passed by the Termination routine is X'04', it means that processing stopped while the last step of the job was being terminated. Entry to the Termination routine from the Restart Determination routine was a "normal" entry; the Restart Preparation routine was not executed, and job termination has been completed. The Restart Determination routine passes the X'04' return code to module IEFSD305.

When control is returned to module IEFSD305, the routine inspects the return code passed to it by the Restart Determination routine. It uses the WTO macro instruction to inform the operator of the job name, the step name, the procedure step name, and the job status, then processes the entry.

A return code of X'00' indicates the presence of one of two conditions: Either the step was being executed when processing stopped (and is not to be restarted), or the step was being terminated in flush mode, and is not the last step of the job. In either case, module IEFSD305 reads the job's SCD into main storage and enqueues the output queue entries in the appropriate queues. Then, the routine deletes the input queue entry.

A return code of X'04' indicates that processing stopped while the last step of the job was being terminated. In this case, the Termination routine has completed step and job termination processing, and module IEFSD305 need only delete the job's input queue entry.

A return code of X'08' indicates that the job is to be restarted. If the step was being executed when processing stopped, the SCT pointer in the JCT points to the SCT for the current step. The Restart Preparation routine has determined that a restart is possible, has been requested, and has been authorized. Module IEFSD305 deletes the job name from the jobnames table, adds it to the restart activation jobnames table, and enqueues the job's entry in the hold queue.

If the step was being terminated when processing stopped, the SCT pointer in the JCT points to the SCT for the next step. The step has been normally terminated, and module IEFSD305 deletes the job name from the jobnames table and enqueues the job's entry in the appropriate input queue at priority 14.

A return code of X'0C' indicates that an I/O error prevented the TIOT Read/Write routine from reading the job step TIOT. In this case, the step could not be processed by the Termination routine and cannot be restarted. Module IEFSD305 reads the last Allocation SMB, reads and verifies the validity of each record in the SMB chain, and updates the last valid SMB to point to the first Interpreter SMB for the next step. Then the routine reads the SCD, uses it to enqueue the job's system output queue entries, and deletes the input queue entry.

A return code of X'10' indicates that processing stopped before the step started execution. Since the job step TIOT was not saved (if it has been constructed) the job cannot be restarted. Module IEFSD305 updates the last Allocation SMB to point to the first Interpreter SMB for the next step, then reads each DSB and uses the JFCB pointer to read a record from the queue data set.

If the record is not a valid JFCB, module IEFSD305 sets the DSB to zero (to indicate that it is the last in the chain) and writes the DSB back into the queue data set. Next it reads the SCD and enqueues the job's output queue entries, then deletes the job's input queue entry.

A return code of X'14' indicates that the system conversion routine, IECPCNVT, located in the nucleus, has stopped the termination processing for the job being executed. This action occurs in the following situation: During job termination processing, the termination routine in module IEFSD42Q may request the nucleus conversion routine to change a relative track (TTR) value to an absolute (MBBCCHHR) value. If module IEFSD42Q passes an invalid TTR value (e.g., one that is all zeros, all blanks, or not on the SYS1.SYSJOBQE data set) to the conversion routine, an abnormal termination occurs. Control returns to module IEFVSDRD, which gives the return code (X'14') to module IEFSD305. Module IEFSD305 stops the job and prints a message.

Dequeued RJE Queue Entries

The entry is dequeued from the purge queue and re-enqueued on the RJE queue with a priority of 14. If the corresponding input queue entry has not been enqueued, the LTH for the RJE queue entry is modified appropriately.

Dequeued ASB Queue Entries

The entry is dequeued from the purge queue and re-enqueued on the ASB queue with a priority of 14.

Dequeued Hold Queue Entries

The entry is dequeued from the purge queue and re-enqueued in the hold queue.

When the processing of the purge queue is completed, module IEFSD305 tests the restart activation jobnames table. If there is at least one entry, the routine branches to the Restart Activation routine (module IEFVSDRA). This routine constructs a command to start the Restart Reader. The command, which starts the procedure IEFREINT, may contain up to 13 jobnames specifying jobs to be restarted. If more than 13 jobs are to be restarted, the routine constructs an additional START command. When it has constructed the START commands, the Restart Activation routine issues the MGCR macro instruction to schedule the execution of the commands, then branches back to module IEFSD305.

Module IEFSD305 determines the jobs that are to be restarted and saves the corresponding jobnames from the initiator jobname table (see Figure 6). The module then zeros these jobname entries in the initiator jobname table and compresses the consolidated jobnames table (CJT), which consists of the interpreter jobnames table and the initiator jobnames table, to remove the zeroed entries. The remaining names in the CJT correspond to jobs that are not to be restarted and whose data sets, therefore, are to be scratched.

Module IEFSD305 then branches to the Scratch Data Set utility (module IEFSD304) to scratch all data sets with generated names (except for system output data sets) created for these jobs.

Scratching Data Sets

When module IEFSD304 is entered, it searches the system UCBs and makes a list of those corresponding to direct access devices (except data cell drives) in the ready state. When it has completed the list, module IEFSD304 uses the ATTACH macro instruction to pass control to the Scratch Data Set Utility routine.

The scratch data set utility routine (module IEFSD308) reads each DSCB in the VTOC of the volume mounted on the indicated unit. When it finds a system output data set name, it adds a one to the user count field in the UCB for that volume. When it finds a system input data set name, it compares the fourth field in the name to each of the job names in the consolidated jobnames table. If it finds a match, it scratches the data set; if not, it reads the next DSCB. When it has processed all DSCBs in the VTOC, the Scratch Data Set Utility routine returns control to module IEFSD304, which continues its search of the UCBs.

Since the scratch data set utility processes only one device before returning control to its caller, an ATTACH macro instruction corresponding to each UCB in the list must be issued. If there are six or less UCBs in the list, the required number of ATTACH macro instructions are issued, followed by a WAIT macro instruction. If however, there are more than six UCBs in the list, six ATTACH macro instructions (and a WAIT macro instruction) are issued initially; the remaining tasks are attached as ECBs are posted.

The devices are processed in an order that optimizes channel activity. The six initial tasks are to process one device on each channel having a direct access device in the ready state, in ascending order by channel. If additional devices must be processed, they are processed in order of channel activity.

When all devices on the list have been processed, control is returned to module IEFSD055, which passes control to module IEFQINTZ. Module IEFQINTZ releases the region, and returns control to the Master Scheduler IPL routine.

Part 2: Processing Input Streams

An input stream is a series of records that primarily contains job control language (JCL) statements; it may also contain references to procedures cataloged in the procedure library, data sets to be processed by job steps, and operator commands. The information in an input stream, and the information to which the input stream refers, controls the job processing performed by the operating system.

In order to use the information in an input stream, the operating system must read (obtain) input stream records and then interpret them (convert them to tabular format). The reading and interpreting functions may be performed as parts of the same system task, or they may be performed by routines executing separate tasks. This section describes several methods of performing the reading function; the interpreting function, which is associated with reading, is always performed by the Interpreter subroutine, and is discussed in the "Common Elements" section of this publication.

The Reading Task

The first task that must be accomplished by the operating system in order to process jobs is the reading task. A reading task is established each time an appropriate START command is issued. The task performed by a reader is to read input stream records until it encounters an end-of-data condition or a STOP command. When the routine performing the reading task encounters a JCL statement, it stores the statement in an internal queue. When it encounters data to be used as input to a program performing a job step, the reader places the data on a direct access volume and generates a DD statement to replace the DD * statement in the input stream. There are several ways to accomplish the reading task:

- The Interpreter subroutine may be used as a direct reader. In this case, both the reading and interpreting functions are performed as parts of the same system task: the interpreter reads records from the specified input stream, converts them to tabular form, and stores them in an input queue.
- The Automatic SYSIN Batching (ASB) routine may be used to read the input stream. In this case, the reading and interpreting functions are performed as separate tasks: the ASB routine reads the input stream, stores the JCL statements for several jobs in a batching queue, and establishes a temporary task to perform the interpreting function.
- The Remote Job Entry (RJE) routines may be used to read the input stream. In this case, the RJE routines read input stream records from remote locations, store them in an intermediate queue, and the Interpreter subroutine uses the intermediate queue as its input source. (The RJE routines are described in detail in the RJE Program Logic Manual.)

USING THE INTERPRETER AS A READER

When a START RDR command is issued, it causes the System Task Control routine to initiate the operation of the Interpreter Reader Control routine (module IEFIRC) in the problem program mode. In order to provide flexibility in the assignment of an I/O device to the input stream, this routine is initiated and terminated in a manner similar to the way a job is processed. The initiating function is performed by the System Task Control routine on the basis of information furnished by the operator in the START command, and information contained in the procedure library (in the RDR procedure).

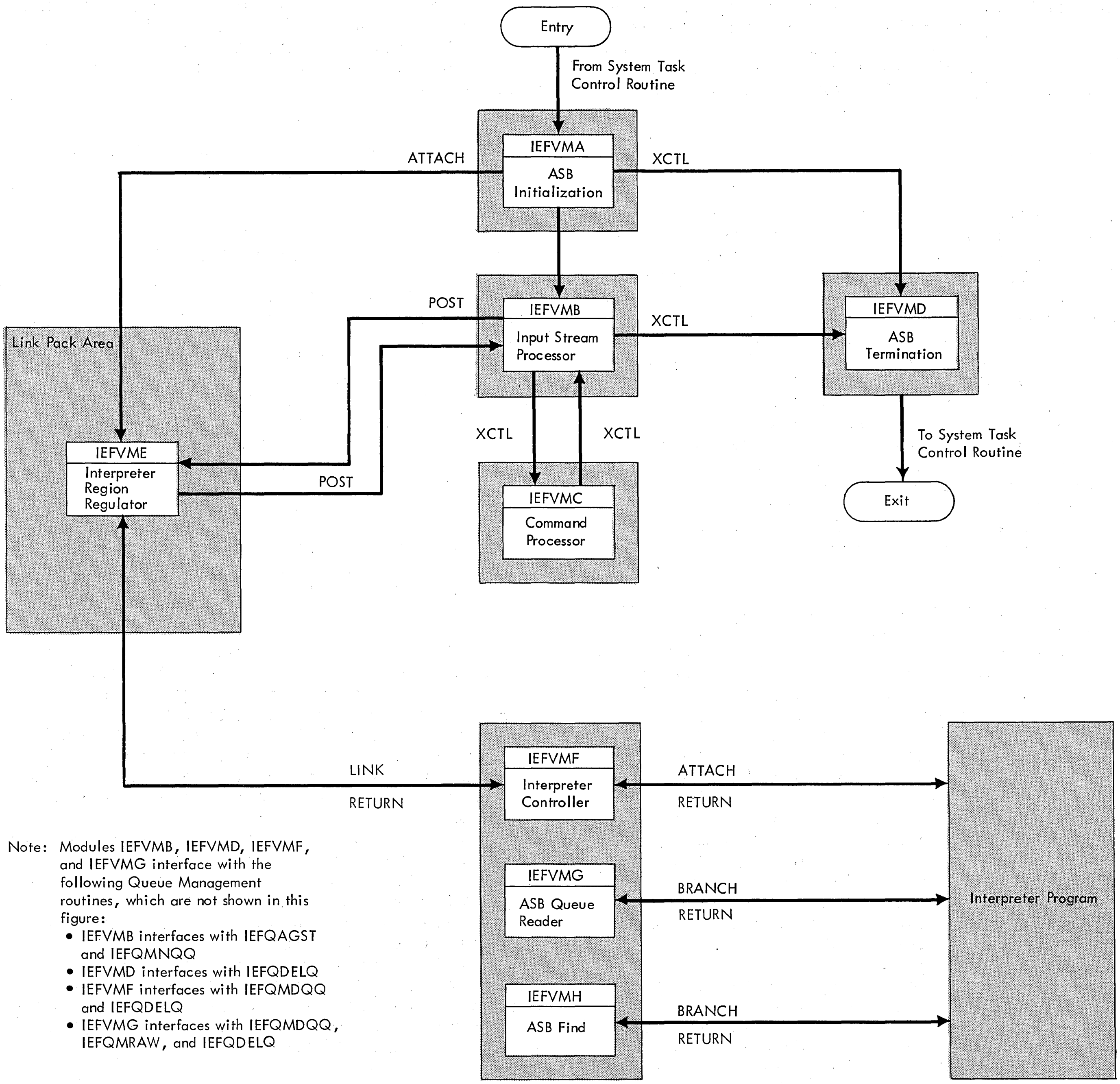
The System Task Control routine (which is described in detail in the "Command Processing" section of this publication) receives control via an ATTACH macro instruction issued in the Master Scheduler Attach routine. The System Task Control routine builds an internal data set of JCL statements created from the information in the START command, then uses the Interpreter subroutine to combine these JCL statements with the statements in the RDR procedure, and to convert the information to tabular format. It uses the Initiator subroutine, which invokes the I/O Device Allocation routine (see the "Common Elements" section of this publication) to assign an I/O device to the input stream, obtains a region, and passes control to the Interpreter Reader Control routine via an ATTACH macro instruction.

The Interpreter Reader Control routine acquires storage for and builds the interpreter entrance list (NEL), then uses the LINK macro instruction to pass control to the Interpreter subroutine. The interpreter processes the input stream until a STOP command is received or the input stream is exhausted, then returns control to the Interpreter Reader Control routine. The Interpreter Reader Control routine returns control to the Initiator subroutine, which uses the Termination routine (also described in the "Common Elements" section of this publication) to terminate the task.

The Initiator subroutine then returns control to the System Task Control routine, which frees the region and returns control to the Master scheduler.

USING THE ASB ROUTINE AS A READER

The issuing of a START RDRA command causes the System Task Control routine to initiate operation of the ASB program (Figure 7) in the same way it initiates operation of the Interpreter Reader Control program.



Note: Modules IEFVMB, IEFVMD, IEFVMF, and IEFVMG interface with the following Queue Management routines, which are not shown in this figure:

- IEFVMB interfaces with IEFQAGST and IEFQMNQQ
- IEFVMD interfaces with IEFQDELQ
- IEFVMF interfaces with IEFQMDQQ and IEFQDELQ
- IEFVMG interfaces with IEFQMDQQ, IEFQMRAW, and IEFQDELQ

Figure 7. ASB Routine

ASB Control Flow

Control is initially passed to the ASB Initialization routine (module IEFVMA), which acquires main storage for the ASB work area (ASBWA), validates and saves the values from the PARM field buffer passed by the Initiator subroutine, uses the EXTRACT macro instruction to access the communications parameter area (CPA), and initializes data set processing. (The ASBWA is shown in Figure 8.)

The ASB Initialization routine also uses an ATTACH macro instruction to initialize the Interpreter Region Regulator routine (module IEFVME), which is resident in the link pack area. The ASB Initialization routine passes a five-word parameter list to the Interpreter Region Regulator routine. The parameter list contains a pointer to the PARM field buffer, a pointer to the ID of the console that issued the START command, a fullword containing the subpool ID and the region size of the region required for the interpreter, and two ECBs used for communication between the Interpreter Region Regulator routine and the Input Stream Processor routine. The Interpreter Region Regulator routine immediately issues a WAIT macro instruction using the first of the communication ECBs. After attaching the Interpreter Region Regulator routine, the ASB Initialization routine passes control to the Input Stream Processor routine via an XCTL macro instruction.

Offset Hex	Offset Dec										
0	0	Address of PARM Field from Started Procedure	4	Address of Console ID from START CIB	4						
8	8	Subpool and Size of Interpreter Region	4	Start Interpreter ECB	4						
10	16	Interpreter Complete ECB	4	Address of Interpreter Region Regulator TCB	4						
18	24	Address of Communications Parameter Area (CPA)	4	Address of Current Input Statement	4						
20	32	Address of Procedure Library Input Buffer	4	Address of Next Procedure Statement in Buffer	4						
28	40	Command Authority	2	TTR of IEFDATA JFCB	3	Sw A	1	Sw B	1	Sw C	1
30	48	Sw D	1	Sw E	1	WTO Message Buffer	55				
68	104			Command Disp	1	Termination Message Number	1	Console ID	1	General Work Area or DDname Save Area	19
70	112										
78	120									Reserved	1
80	128	Number of Spooled Data Records	4	Reserved	4						
88	136	IEFDATA DCB End-of-Volume Exit	4	IEFDATA DCB In-Storage JFCB Exit	4						
90	144	Batching Device Type	8								
98	152	Name on Last Procedure's EXECUTE Statement	8								
A0	160	Stepname on Last DD Override	8								
A8	168	Name on Last DD Statement	8								
B0	176	Maximum BLKSIZE	2	Max BUFNO	1	Reserved	1	Number of Jobs Per Batch	2	Number of Queue Tracks Available	2
B8	184	Number of Queue Tracks Used	2	Current Number of Jobs in Batch	2	Reserved	4				

Figure 8. Automatic SYSIN Batching Work Area (ASBWA) (Part 1 of 2)

Offset Hex	Offset Dec								
C0	192	General Work Area	8						
C8	200	Address of IEFDATA TIOT	4	QMPA	36				
D0	208	QMPA							
F0	240	Record Buffer Address	4	Queue TTR Pointer	4				
F8	248	Third TTR Pointer	4	Address of Queue Record	4				
100	256	Address of Next Space in Queue Record	4	Block Address from Compress/Decompress Parm List	4				
108	264	Block Length from Comp/Decomp Parm List	2	Reserved	2	Serial Number for Unique DSNAME	4		
110	272	Serial Number At Start of Current Job	4	Address of Verb in Current JCL Statement	4				
118	280	Address of Operand in Current JCL Statement	4	TTR of Last DD* or DD DATA Queue Record	4				
120	288	TTR of Procedure Library Member	4	Reserved (7094 Emulator Only)	4				
128	296	Pointer to Binary SYSIN Workarea	4	DD Statement Name Length	2	Input Stmt Name Length	2		
130	304	Address of Compression/Decompression Parameter List	4	Address of Record to be Compressed	4				
138	312	Address Where Compressed Record is to be Written	4	Length of Rec to be Compressed	2	Length of Output Block	2		
140	320	Char to be Removed	1	Compression Switch	1	Reserved	2	IEFRDER DCB	96
148	328	IEFRDER DCB		IEFPDSI DCB	88				
1A8	424	IEFPDSI DCB		IEFDATA DCB	96				
200	512	IEFDATA DCB		Address of JFCB	4				
260	608	Address of Next Entry in In-stream Procedure Directory	4	Address of In-stream Procedure Directory	4				
268	616	Character Delimiter Value to Replace /*	4						

Figure 8. Automatic SYSIN Batching Work Area (ASBWA) (Part 2 of 2)

The Input Stream Processor routine (module IEFVMB) reads the input stream. It places the JCL statements in the ASB queue in a special compressed format and writes any system input data sets onto a direct access volume. After the data is written, a DD statement describing the direct access data set is generated and written in place of the original SYSIN DD statement in the ASB queue. If a PROC statement is read, the routine saves the name field in the in-stream procedure directory (located in the in-stream procedure work area) for the job. All JCL following the PROC statement is then written directly to the ASB queue until either a PEND statement ending the in-stream procedure is read or the next JOB statement is read. If an EXEC statement that contains a procedure name is read, the routine searches the in-stream procedure directory for the specified procedure name. If the routine finds the procedure in the in-stream procedure directory, the EXEC statement is written to the ASB queue without further processing. However, if the procedure was not an in-stream procedure, the Input Stream Processor routine issues a BLDL macro instruction to locate the procedure in the procedure library. It then generates a special statement identifying the procedure (including the relative track address of the procedure's first record in the procedure library) and writes it in the ASB queue immediately preceding the EXEC statement.

Whenever the Input Stream Processor routine encounters an unrecognizable statement, it passes control to the Command Processor routine (module IEFVMC) via an XCTL macro instruction. This routine determines whether the statement is a command verb. If it is a command verb, theroutine processes it according to the disposition specified in the RDRA procedure. If it is not a command verb, a switch is set in the ASBWA informing the Input Stream Processor routine of the error. In either case, control is returned to the Input Stream Processor routine. If the error switch is on in the ASBWA, the Input Stream Processor routine writes the statement in the ASB queue for subsequent error diagnosis by the Interpreter.

The Input Stream Processor routine and the ASB Initialization routine use the ASB Termination routine (module IEFVMD) to handle both normal and error termination conditions. This routine resets the system resources used by the ASB routine to their original status, and generates an error message if termination was caused by an error.

The input stream processing continues until one of the following conditions arises:

- The number of jobs specified in the RDRA procedure has been read and placed in the ASB queue.

- The input stream has been exhausted.

- The number of tracks allocated for the ASB queue (specified in the RDRA procedure) has been exhasted or a STOP command has been encountered.

- At least one complete job has been placed in the ASB queue and the ASB routine cannot allocate more direct access space for copying SYSIN data sets.

When one of these conditions occurs, the Input Stream Processor routine initiates interpretation by issuing a POST macro instruction for the communications ECB for which the Interpreter Region Regulator routine is waiting. The Input Stream Processor routine then continues to process one additional job before issuing a WAIT macro instruction using the other communications ECB which it shares with Interpreter Region Regulator routine. Although the records for the additional job are placed in the ASB queue, they are not actually enqueued (by placing a pointer in the QCR) until after interpretation has been completed.

The Interpreter Region Regulator routine issues a request for a main storage region in which to execute the Interpreter program. When such a region becomes available, the routine passes control to the Interpreter Controller routine (module IEFVMF) via a LINK macro instruction.

The Interpreter Controller routine constructs the Interpreter entrance list (NEL). Special access methods to be used by the Interpreter for both reading JCL statements and finding cataloged procedures are indicated by entries in the NEL exit list. NEL entries are also set up to establish communication between the Interpreter Subroutine and the special access method routines, which are control sections in the Interpreter Controller routine. After constructing the NEL, the Interpreter Controller routine enters the Interpreter program via an ATTACH macro instruction and waits for Interpreter completion.

The NEL exit list entry indicates to the Interpreter subroutine that the ASB Queue Reader routine (module IEFVMG) should be entered whenever a GET is issued to obtain input stream records. The ASB Queue Reader routine reads and expands a compressed JCL statement from the ASB queue. After the expanded statement is built in the Special Access Method Work Area (SAMWA), it is passed to the Interpreter Subroutine for processing. (The SAMWA is shown in Figure 9.)

When the ASB Queue Reader routine encounters the special statement generated by the Input Stream Processor routine to indicate a cataloged procedure, the statement is used to fill the procedure name and first record TTR fields in the SAMWA, which is shared by the ASB Queue Reader routine and the ASB Find routine (module IEFVMH). The ASB Queue Reader routine then processes the next statement in the ASB queue, which is the execute statement for the cataloged procedure in question. Using the NEL exit list entry, the Interpreter subroutine branches to the ASB Find routine instead of issuing a FIND macro instruction to locate the procedure. The ASB Find routine uses the information in the SAMWA to return the information normally returned to the Interpreter subroutine after execution of a FIND macro instruction. The special FIND macro instruction processing removes the relatively slow FIND operation from the Interpreter subroutine and places it in the

Offset Hex	Offset Dec				
0	0	QMPA 36			
24	36	Address of Queue Record Buffer 4		Current Queue Record TTR 4	
2C	44	Current Queue Record 176			
DC	220	JCL Statement Buffer 80			
12C	300	Reserved 4		Dequeue ECB	
134	308	Continued 8		Address of Queue Mgmt Control Routine 4	
13C	316	Address of Queue Mgmt Delete Routine 4		Address of Queue Management Unchain Routine 4	
144	324	I/O Error Retry Count 2	Reserved 2	Address of Queue Management Dequeue Routine 4	
14C	332	Track Stacking Parameter List 4		72	
194	404	Queue Management Save Area			
				In-Stream Procedure Name	
19C	412	Continued 8		TTR of First Record in In-Stream Procedure 4	
1A4	420	Address of Compression/Decompression Parameter List 4		Address of Decompressed Record Buffer 4	
1AC	428	Address of Block Containing Compressed Records 4		Length of Decompressed Record 2	Length of Block Containing Compressed Records 2
1B4	436	Character Removed 1	Compression Switches 1	Reserved for Record Decompression Routine 2	

Figure 9. Special Access Method Work Area (SAMWA)

Input Stream Processor routine, which requires much less main storage.

Upon return from the Interpreter program, the Interpreter Controller routine checks the return code for an I/O error, a 'queue full' condition, or an Interpreter ABEND.

Normally, the Queue Management routines wait for space to become available in the queue when a 'queue full' condition is encountered; however, when the ASB routine is being used, a switch is set in the queue management parameter area (QMPA) which causes the Queue Management routines to pass a return code when such a condition arises. When the Interpreter program recognizes this code, it purges any input or output queue entries created for the job and terminates processing.

If either a 'queue full' or I/O error condition was indicated to the Interpreter Controller routine, the routine reconstructs the job control language statements for the job being interpreted when the condition arose and places them back in the ASB queue. If Interpreter ABEND code was indicated, an error message is generated by the Interpreter Controller routine and the remaining entries in the ASB queue are purged. The Interpreter program is restarted after the first occurrence of an I/O error. In all other cases, control is returned to the Interpreter Region Regulator routine.

The Interpreter Region Regulator routine normally posts the completion of its task (using the shared communication ECB), and thus returns control to the Input Stream Processor routine with the return code from the Interpreter program. However, if a 'queue full' condition was indicated, it issues an ENQ macro instruction to obtain exclusive control of the no-queue-space ECB. Next the routine clears the ECB to remove the POST that occurred when the interpreter purged the input and output queue entries for the job. It then issues a WAIT macro instruction using the no-queue-space ECB to allow job terminations to free additional queue space. When the ECB is posted, the routine issues a DEQ to release control of the no-queue-space ECB, acquires a new region of main storage, and restarts the Interperpreter Controller routine. After completion, the routine again issues a WAIT macro instruction using the communication ECB.

Upon return from the Interpreter Region Regulator routine, the Input Stream Processor routine normally restarts the reading of the input stream. However, if there is no more input to be read, or if the return code from the POST macro instruction indicated the occurrence of an error requiring termination, the routine passes control to the ASB Termination routine, which terminates the task.

ASB I/O Exceptions

The SYSIN batching function involves the use of four data sets. They are:

- The input stream, which is read by the Input Stream Processor routine.
- The compressed JCL stream (ASB queue), which is written by the Input Stream Processor routine and read by the ASB Queue Reader routine.
- The system input data sets, which are written on direct access volumes by the Input Stream Processor routine.
- The procedure library, which is read by the Input Stream Processor routine.

Exceptional I/O conditions -- read or write errors and space availability problems -- are handled in most cases by deleting all in progress queue entries and scratching all data sets created for the job. The Interpreter subroutine is initiated for any jobs completed in the ASB queue and processing is terminated after freeing the ASB work area.

ASB Job Control Language Compression

To make efficient use of available queue space, the ASB routine compresses JCL statements before placing them in the ASB queue. To perform this compression, the Input Stream Processor passes control to the Record Compression routine (module IEZNCODE). The Record Compression routine compresses statements by replacing contiguous occurrences of a specified character with a count field. A detailed description of the routine can be found in Part 6 of this publication under the section entitled "The Interpreter Routine: Auxiliary Routines."

When the ASB Queue routine receives control to access jobs in the ASB queue, it uses the Record Decompression routine (module IEZDCODE) to decompress the compressed statements. A description of the Record Decompression routine can also be found in the section noted above.

ASB Queue Processing

The ASB routine makes extensive use of Queue Management routines to store compressed JCL statements in the ASB queue and later retrieve them for use as input to the Interpreter program. Each job is enqueued as a separate string of forward-chained 176-byte queue records, each record containing as many compressed JCL stataments as possible. The end of a job being retrieved from the ASB queue is recognized by a forward pointer of all blanks.

The Queue Management routines that are used to accomplish the storage and retrieval functions are described in detail in the "Common Elements of Job Management" section of this publication. The ways in which Queue Management routines are used by ASB routines are described in this section.

There are four ASB routines that use Queue Management routines: the Input Stream Processor routine, the ASB Queue Reader routine, the ASB Termination routine, and the Interpreter Controller routine.

Input Stream Processor Routine

At the start of each job, the Input Stream Processor routine uses the Queue Management Assign/Start routine to initialize the QMPA for ASB queue processing and to assign space for three queue records. The first two queue records are used as the beginning of the compressed JCL chain. The third space is used to contain a JFCB for the system input data sets, if any are included in the job. The queue address of the third space is moved to the TIOT so the JFCB will be written there when the data set is opened.

The initial assignment of two queue records is necessary to establish the forward chain. A pointer to the second record is placed in the first four bytes of the first record before the first record is written.

As 176-byte queue records are filled with compressed JCL statements, the Input Stream Processor routine uses the Queue Management Read/Write routine to place them on a direct access volume and assign space for the next queue record. In order to maintain the forward chain, the Input Stream Processor routine stays one record ahead of the current record being written.

When the Input Stream Processor routine recognizes the first statement of the next job, the forward pointer in the current queue record to be written is set to all blanks to indicate the end of the chain. The routine then uses the Queue Management Read/Write routine to place the last record on a direct access volume and uses the Queue Management Enqueue routine to enqueue the job at priority seven. (All jobs are enqueued at priority seven so that the first job placed in the queue will be the first job retrieved for Interpreter processing.)

ASB Queue Reader Routine

The ASB Queue Reader routine uses the Queue Management Dequeue routine to access jobs in the ASB queue. Since all jobs have the same priority, the first job enqueued will be the first job dequeued. Subsequent records in the job's chain are read by the Queue Management Read/Write routine using the forward pointer in the previous record.

When a forward pointer of all blanks is encountered, the ASB Queue Reader routine uses the Queue Management Delete routine to delete the job's entry from the ASB queue. The logical tracks used by the job's entry are returned to the free track queue, and the ASB Queue Reader routine uses the Queue Management Dequeue routine to access the next job in the ASB queue.

When the ASB Queue Reader routine enters the Queue Management Dequeue routine and the ASB Queue is empty, a 'no-work' condition exists. This condition results in the insertion of an ECB in the no-work ECB chain. (This ECB is posted when work becomes available.) Instead of waiting for work to become available, the ASB Queue Reader routine uses the Queue Management Unchain routine to remove the ECB from the no-work ECB chain and then causes termination of the Interpreter program.

ASB Termination Routine

If the ASB Termination routine is entered because of an error condition, and if a job has been partially entered in the ASB queue, the routine uses the Queue Management Delete routine to delete the partially completed job from the ASB queue, thus returning the queue space assigned to the job to the free track queue.

Interpreter Controller Routine

The Interpreter Controller routine uses Queue Management routines when error conditions arise that necessitate changes in the ASB queue.

When the first I/O error condition or when a queue full condition arises, the Interpreter Controller routine reenqueues the current job being interpreted. Since the forward chaining of the job's queue records has not been changed, this returns the job to its condition before interpretation was initiated.

When an error arises during interpretation which cannot be handled by restarting the Interpreter program, the Interpreter Controller routine use the Queue Management Delete routine to remove the job from the ASB queue.

If an error condition occurs that will result in termination of the ASB routine, the Interpreter Controller routine uses the Queue Management Dequeue routine and the Queue Management Delete routine to remove all remaining jobs from the ASB queue.

ASB SYSIN Data Set Processing

Most of the initialization for the writing of SYSIN data on direct access volumes is accomplished in the ASB Initialization routine. The routine does the following:

- Locates the SYSIN data set (IEFDATA) entry in the RDRA procedure TIOT.
- Stores the queue address of the JFCB for SYSIN data sets in the ASBWA.
- Reads the JFCB for SYSIN data sets into the ASBWA.
- Sets bits in the UCBs pointed to in the IEFDATA device entries in the TIOT indicating that SYSIN data set devices, which are not already specified as permanently resident or reserved, are reserved and public.
- Constructs, in the SYSIN data set JFCB in the ASBWA, the base of a unique data set name to be generated for SYSIN data sets.
- Stores the blocksize and buffer number values in the SYSIN data set JFCB in another area of the ASBWA.

The constructed base of the unique data set name is:

SYSddddd.Tttttt.IV000.

The 'ddddd' represents a five-byte current date field, and the 'tttttt' represents a six-byte current time field.

The Input Stream Processor routine performs one initialization function. It sets up entries in the SYSIN data set DCB that allow the Input Stream Processor routine to process end-of-volume conditions and that indicate processing will take place with a JFCB which is already in main storage (not in a queue).

Checking DD * and DD DATA Statements

When the Input Stream Processor routine encounters a DD * or DD DATA statement, it searches for, and checks the syntax of the value of, the DLM parameter. If the parameter's value is valid, the routine stores it in the last field shown for the ASBWA (Figure 8). Then the routine searches for the BLKSIZE and BUFNO subparameters of the DCB parameter, and if present, stores them in the SYSIN data set JFCB in the ASBWA.

If either the block size or buffer number subparameters are not specified in the DD statement in the input stream, the Input Stream Processor routine checks to see if the job step that includes the SYSIN data set specifies the execution of a cataloged procedure. If a cataloged procedure is indicated, the Input Stream Processor routine locates the procedure and searches it for a DD statement with a DDNAME parameter that matches the name of the DD * or DD DATA statement in the input stream. If such a statement is found, the routine uses the block size and buffer number subparameters from the cataloged statement to take the place of whichever subparameter is missing (or both, if both are missing) on the input stream statement. If a subparameter cannot be found in either location -- the input stream statement or the procedure statement -- the value specified in the original JFCB entry is restored.

The Input Stream Processor routine then checks the validity of the BLKSIZE and BUFNO subparamenters. Block size must be a multiple of 80 and equal to or less than the block size value specified originally in the SYSIN data set JFCB. The buffer number must be equal to or less than its originally specified value.

Processing DD * and DD DATA Statements

When the Input Stream Processor routine encounters a DD * or DD DATA statement, it saves the address of the queue location where the next record is to be written and the address where the next record will be placed in the current buffer. The DD statement and any continuation statements are then compressed and placed in the queue record and any subsequently needed queue records.

If an error is detected after processing the DD * or DD DATA statement, the SYSIN data set in the input stream is flushed, a /* statement is placed in the queue record following the DD * or DD DATA entry, and processing of the job's input stream records continues. The Interpreter subroutine diagnoses the error and prints the appropriate programmer message.

If no errors are detected in the DD * or DD DATA statement, the Input Stream Processor routine obtains the queue address that was saved when the DD* or DD DATA statement was encountered and sets the address of the next record in the block to point to the location where the DD statement was written. This procedure removes the DD * or DD DATA statement from the queue and allows a generated DD statement describing the direct access volume data set to be put in its place.

The Input Stream Processor routine then constructs the remaining portion of the unique data set name for the SYSIN data set, as follows:

jjjjjjjj.Snnnnnnn

The 'jjjjjjjj' represents the current job name. This field may be from one to eight bytes long. The 'nnnnnnn' represents a seven-byte generated serial number, which, beginning at one, is increased by one for each SYSIN data set written on a direct access volume. The format of the complete unique data set name is:

SYSddddd.Ttttttt.IV000.jjjjjjjj.Snnnnnnn

Using the UCBs pointed to in the IEFDATA device entries in the TIOT, the Input Stream Processor routine attempts to allocate space for the SYSIN data set. If the attempt is unsuccessful, the routine issues a message to the operator and waits for one minute to allow space to become available through job terminations. After completing the wait, the Input Stream Processor routine checks the stop RDRA ECB and terminates operation if so requested by the operator. If termination is not requested, the allocation procedure is repeated until the necessary space is acquired, after which the data set is opened and the input stream data records are written on the direct access volume.

The Input Stream Processor routine then completes the generated DD statement, using the device entries in the TIOT to obtain the number of volumes written to and their volume serial numbers. The routine then compresses the DD statement in the current queue record and continues the processing of the next input stream record.

Restarting Jobs

The execution of a job step may be terminated as a result of a CANCEL command, because the step's timer interval expired, because of a program error, or as a result of a system ABEND. If the job is eligible for restart, and if the operator authorizes the restart, the step will be restarted automatically by the operating system. Otherwise, the programmer may request a deferred restart by coding the RESTART parameter on the JOB statement, adding a SYSCHK DD statement, and resubmitting the job.

Much of the processing required before a job can be restarted automatically is performed in the Termination routine, in the initiator, and in the System Restart routines. This processing results in the job's input queue entry having been re-enqueued in the hold queue, and in the MGCR macro instruction[1] having been issued to schedule the execution of the START Restart Reader command.

If the restart is a deferred restart, the job is resubmitted through the input stream. In the case of a deferred step restart, the interpreter simply points the JCT to the SCT of the step to be restarted (instead of pointing it to the first step); the job is then initiated normally, starting with the restart step. In the case of a deferred checkpoint restart, however, the interpreter generates an EXEC statement and uses it in conjunction with the SYSCHK DD statement submitted by the programmer to create an extra step. The extra step is the DSDR Processing step; it is executed first (by the DSDR Processing routine), and its SCT points to the step to be restarted from a checkpoint.

This section discusses the processing performed in the Restart Reader routine, and also discusses the processing performed in the Data Set Descriptor Record (DSDR) Processing routine, which is executed as the first step of the restarting job when a checkpoint restart is to be performed.

THE RESTART READER

When the Command Scheduling routine (SVC34) encounters a START command, it posts the Master Scheduler. The Master Scheduler passes control (via the ATTACH macro instruction) to the System Task Control routine. The System Task Control routine passes control to the Initiator subroutine, which attaches the system program specified via the command.

If the command is a START Restart Reader (S IEFREINT,,,jobname,...jobname), the program that gains control is the Restart Reader routine.

A START Restart Reader command is constructed and issued when a step is to be restarted automatically: as a result of an abnormal termination during the execution of the step, or as a result of system restart processing. When the Restart Reader (module IEFVRRC) is entered, the job's input queue entry has been partially processed by an initiator while starting the job, and has been re-enqueued in the hold queue. The function of the Restart Reader is to restore the queue entry to its state before the job was initiated, and if a checkpoint restart is planned, to add the control blocks and tables that will cause the DSDR processing routine to be executed as the first step.

The Restart Reader constructs a QMPA, extracts a job name from the restart activation jobnames tables, and uses the Queue Management Dequeue routine to dequeue the job from the hold queue and read the JCT into main storage. The Dequeue routine returns control to the Restart Reader, which uses the XCTL macro instruction to pass control to a linkage routine. The linkage routine (module IEFRCLN1) passes control to the Interpreter Initialization routine via a LINK macro instruction.

The interface between the Restart Reader and the Interpreter is shown in Figure 10.

The Interpreter is used to re-read the original JCL, and to create a queue entry that duplicates the original queue entry. The Interpreter does not, however, actually enqueue the entry.

The original JCL is available, in condensed form, in the SMBs created for the job when its input stream was first processed. The Interpreter uses a special access method, the SMB Reader, to read the job's SMBs and reconstruct the JCL.

The Restart SVC Issuing routine (module IEFRSTRT) is loaded into main storage via the interpreter access method exit. When the Interpreter Get routine issues the GET macro instruction, the Restart SVC Issuing routine is entered at entry point IEFSMR and issues the restart SVC (SVC 52) to execute the SMB Reader.

When it is entered for the first time, the SMB Reader constructs a QMPA, reads in the job's SCTs, and stores the address of the end of the SMB chain.

[1] The MGCR macro instruction issues SVC34.

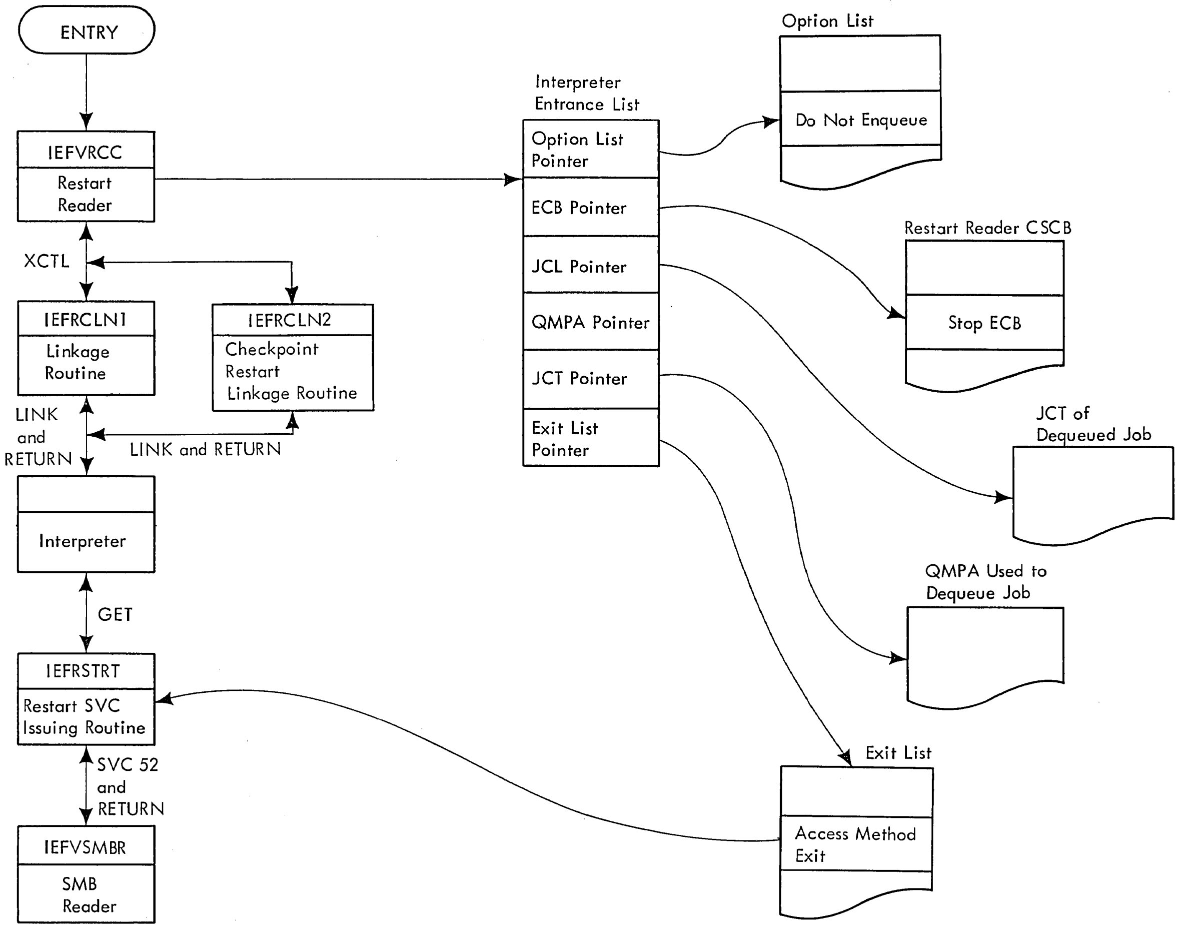


Figure 10. Interface Between the Restart Reader and the Interpreter

Each time the SMB Reader is entered (including the first time) it reads and inspects an SMB. If the SMB contains a message, a DSB entry, or a procedure library statement, the routine ignores it and reads the next SMB in the chain. If the SMB contains a JCL statement, the routine expands it (except that it blanks out the asterisk in column 3 of a converted comments statement), and replaces DD* and DD DATA statements with DD DUMMY statements. The routine places the expanded JCL statement in a buffer and returns control to the Restart SVC Issuing routine; the Restart SVC Issuing routine returns control to the Interpreter. When the routine reaches the end of the SMB chain, it uses the DCB end-of-data exit.

The original JCL is reprocessed by the interpreter. It constructs the approriate tables, and when processing is complete, the interpreter returns control to the Restart Reader, and passes it the addresses of the newly created JCT and of the QMPA used in writing the tables into the work queue data set.

The restart reader uses the SCT pointer in the old JCT to read the SCT for the step to be restarted into main storage, then extracts and stores the step name. Next, it reads the newly created SCTs in order, until it finds the one that corresponds to the step to be restarted.

When it finds the newly created SCT that corresponds to the step to be restarted, the Restart Reader updates the original SCT with the information from the new SCT (except for the queue addresses and dependency codes) and writes the old SCT back into the queue data set.

Using the queue addresses in the old and new SCTs, the Restart Reader updates the other tables for the step to be restarted, writes them back into the queue data set, and then deletes the newly created input and system output queue entries.

The job's queue is now exactly as it was when the step was originally selected for initiation.

If the job is to be restarted from a checkpoint, however, an additional job step (the DSDR Processor) will be executed to prepare for the checkpoint restart, and additional tables must therefore be added to the job's input queue entry. The Restart Reader creates the appropriate JCL statements in main storage. It passes control to a linkage routine (module IEFRCLN2), which passes control to the Interpreter (using the same interface used by the System Task Control routine).

When the Interpreter returns control to the Restart Reader, it passes the addresses of the JCT and the QMPA it used. The Restart Reader extracts the address of the SCT from the JCT, and updates the restarting job's JCT to point to the SCT of the DSDR Processor step. It adjusts the SCT of the DSDR Processor step to point to the SCT of the step to be restarted, then deletes the system output queue entries created by the interpreter for the DSDR step.

When the queue entry has been enqueued, the Restart Reader determines whether there are any unprocessed jobs specified in the restart activation jobnames table. If so, it processes each remaining job. If there are no unprocessed jobs, the Restart Reader returns control to the initiator, which uses the Termination routine to terminate the Restart Reader, then returns control to the System Task Control routine.

RESTARTING THE STEP

If the restart is a step restart, no further special processing is required. When the Restart Reader has completed its processing, the job's input queue entry is in the same form it was in when the step to be restarted was ready to be selected for processing by an initiator for the first time. When the initiator dequeues the job, it will select the restart step as the first step (because the JCT points to the SCT for the restart step) and normal processing will continue.

If the restart is a checkpoint restart, additional processing must be performed. The job's queue entry is in the same form as it was when the step was ready to be initiated (except that the DSDR Processing step has been inserted and will be selected first) but processing will resume at a checkpoint instead of at the beginning of the step. Thus, the queue entry must be made to appear as it did when the checkpoint was taken.

The information that enables the queue entry to be updated is saved when the checkpoint is taken. The DSDR Processing routine is initiated in place of the restart step; it updates the queue entry from the information in the checkpoint record, then changes the name of the program specified to execute the restart step to IEFRSTRT. IEFRSTRT, a program that issues SVC 52[1], is thus executed when the restart step is initiated.

When the DSDR Processing routine (IEFDSDRP) is entered, it finds the initiator's QMPA, reads the JCT to determine the type of restart to be performed, and obtains the JFCB for the checkpoint data set. If a programmer-deferred restart is to be performed, the DSDR Processing step JFCB is the proper JFCB; if an automatic restart is to be performed, the proper JFCB is the one created for the restart step.

The DSDR Processing routine constructs a DCB for the checkpoint data set, opens it, then reads the SCT for the restart step. It changes the program name in the SCT for the restart step to IEFRSTRT, and determines the number of SIOTs in the restart step.

The routine uses the number of SIOTs to calculate the size of the SIOT Processing table (SIOP), which is uses to reduce the number of accesses to the queue data set during DSDR processing. The SIOP has a 15-byte entry for each SIOT; each entry (constructed when the SIOT is read) contains the addresses of the SIOT and its associated JFCB, the ddname, and a byte used to indicate whether a matching DSDR has been found.

When the routine has performed the processing described above, it reads the first SIOT and constructs an SIOP entry, then reads the checkpoint data set header record and uses the information it contains to calculate the beginning addresses and lengths of the area in each storage hierarchy allocated to the restart step. This information is stored in the SCT and used in obtaining a region of main storage when the restart step is initiated.

Next, the routine begins the actual DSDR processing. It reads a DSDR, and attempts to match it with an entry in the SIOP table by comparing ddnames. If no match is found, the routine reads the next SIOT, constructs an SIOP entry, and checks for a match. This process is repeated each time the routine reads a new DSDR. When it finds a match, the routine checks for a blank ddname. The SIOTs (and corresponding

[1] For a discussion of SVC 52 (Restart) and SVC 63 (Checkpoint), see the MVT Supervisor PLM.

DSDRs) for all but the first part of a concatenated data set have blank names; when a blank ddname is encountered, the routine checks the preceding SIOP entry, and only if it has been matched is there a match on the current DSDR.

If the DSDR corresponds to a JFCB (rather than to a GDG bias count table), the routine reads the corresponding JFCB and overlays all non-volume information with the information in the DSDR, except in the following cases:

- If the DSDR corresponds to a DD DUMMY statement, it is ignored; processing continue with the next DSDR.
- If the GDG All bit is on, or if the data set is a system input or system output data set and a deferred restart is being performed, the dsname in the JFCB is retained as well as the volume information. The remaining DSDRs associated with a GDG are not processed.
- If the GDG Single bit is on in the DSDR, it is turned off in the SIOT. The DSDR is otherwise processed normally.

If the DSDR corresponds to a GDG bias count table, and if the restart is a programmer-deferred restart, the corresponding GDG bias count table is overlaid by the DSDR. If the restart is an automatic restart, the records are not processed.

Finally, the DSDR Processing routine reads the first core image record (CIR), and, if the checkpoint data set is on a direct access device, issues the NOTE macro instruction and closes the data set. The routine then writes out the updated JCT, SCT, and SCTX, then returns control to the supervisor.

Part 3: Initiating Tasks

The Initiator subroutine is used to initiate both system tasks and job step tasks. In order to initiate a task, the initiator retrieves job control information about the task and reserves system resources for the use of the task. After the reservations for a task are completed, the initiator passes control to the task. When a task has terminated, the Termination routine of the initiator releases the task's system resources and returns control to the System Task Control routine.

Initiating System Tasks

When the System Task Control routine is processing a starting system task, it uses the initiator as a subroutine to continue the processing. The Initiator Control routine (module IEEVICTL) provides an interface between the System Task Control routine and the Initiator subroutine by passing the JCT and CSCB for the starting task to the initiator. The Initiator subroutine performs region and I/O device allocation, attaches the program that executes the starting task, performs termination functions after execution, and returns control to the System Task Control routine.

Initiating Job Step Tasks

If a START command specifies an initiator, the Initiator subroutine is being invoked to initiate a job step task. In response to the START command specifying the initiator, the Initiator Control routine (module IEEVICTL) serves as the interface between the System Task Control routine and the Initiator subroutine by obtaining storage for and building the initiator entrance list (IEL), the initiator options list, and the initiator exit list (see Appendix A: Tables and Work Areas for a detailed description of these lists). Then the Initiator Control routine passes control to the initiator. The initiator selects one job at a time for processing from the input queues. Each step of a job is processed in turn, and when the last step of a job has ended, the initiator selects another job. This process continues as long as there are jobs in any of the input queues associated with the initiator or until performance of the initiator is stopped by operator command. The Initiator subroutine terminates the task and finally returns control to the System Task Control routine.

GROUPING INITIATORS

Each initiator is capable of selecting jobs from up to eight of the 15 input queues. The START command, or the PARM field of the EXEC statement in the specified initiator procedure, specifies which of the input queues are to be associated with an initiating task. If several START commands are issued, and each specifies the same initiator procedure, a corresponding number of initiators may be started that will process jobs from the same set of input queues.

When this occurs, the initiators are identified as a group by the procedure name used in the START commands. Members may be added to the group by issuing additional START commands; the size of the group may be reduced by issuing STOP commands, and the set of queues associated with the group may be changed by issuing MODIFY commands.

A group may also be defined by using the identifier parameter in the START command. If this is done, the group consists of those initiators with the same procedure name that share an identifier; it is affected by START, STOP, and MODIFY commands containing that identifier.

Communication among the initiators in a group is maintained with the aid of a group control block (GCB). The GCB, which is shown in Figure 11, is constructed by the first member of the group to be established. It is added to a chain of GCBs, where it remains until the last member of the group stops.

<table>
<tr><td colspan="2">Entry Length 2</td><td colspan="2">Reserved 2</td></tr>
<tr><td colspan="4">Pointer to Next GCB in Chain 4</td></tr>
<tr><td colspan="4">Procname 8</td></tr>
<tr><td colspan="4">Identifier 8</td></tr>
<tr><td>Member Count 1</td><td>Stop Count 1</td><td>Modify Count 1</td><td>Reserved 1</td></tr>
<tr><td colspan="4">Pointer to Command Input Buffer 4</td></tr>
</table>

Figure 11. Group Control Block (GCB)

Initiator Functions

The initiator and its subroutines perform the following functions:

- Job Selection: When the System Task Control routine uses the initiator as a subroutine to initiate a system task, the initiator continues processing of the starting system task. However, if the initiator receives control to initiate a job step task, it scans the ECB list to find the first of its queues that contains an entry. If necessary, it waits until an entry becomes available, then dequeues a job and prepares to process the job.

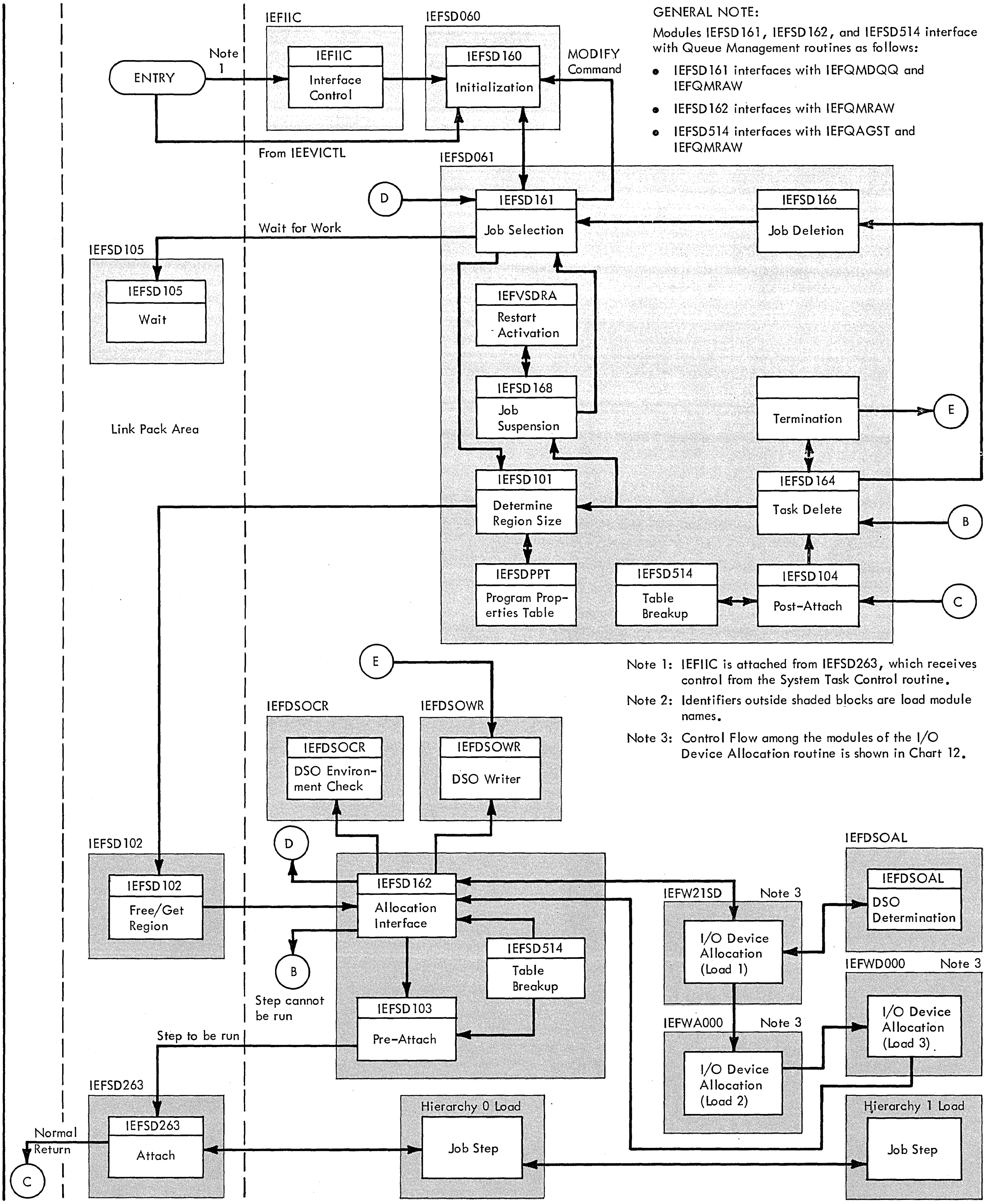
- Region Management: The initiator determines the size of the region required by each task, frees the current region, and obtains a new one of the proper size.

- I/O Device Allocation: The initiator uses the I/O Device Allocation routine (as a subroutine) to check EXEC statement condition codes and to allocate I/O devices to tasks. The I/O Device Allocation routine is described in Part 6 of this publication.

- Attaching Tasks: The initiator gathers information needed by the supervisor to run the task, and transmits this information via the ATTACH macro instruction. Control is passed to the first program of the task; at termination, the supervisor returns control to the Initiator Attach routine.

- Terminating Tasks: The initiator uses the Termination routine (as a subroutine) to release the I/O devices that were allocated to the task, to direct the disposition of data sets used or created by the task, and to check JOB statement condition codes. If the initiator is terminating a job step task and the terminating step is the last step of a job, the initiator uses the Queue Management routines to enqueue the job's system output queue entries, and to delete the job's entry from the input queue. The Termination routine is described in Part 6 of this publication.

Figure 12 shows the flow of control among the modules of the initiator and indicates the sequence in which the functions are performed. In Figure 12, the modules are grouped into load modules.



GENERAL NOTE:

Modules IEFSD161, IEFSD162, and IEFSD514 interface with Queue Management routines as follows:

- IEFSD161 interfaces with IEFQMDQQ and IEFQMRAW
- IEFSD162 interfaces with IEFQMRAW
- IEFSD514 interfaces with IEFQAGST and IEFQMRAW

Note 1: IEFIIC is attached from IEFSD263, which receives control from the System Task Control routine.

Note 2: Identifiers outside shaded blocks are load module names.

Note 3: Control Flow among the modules of the I/O Device Allocation routine is shown in Chart 12.

Figure 12. Initiator Load Modules and Control Flow

Initializing the Initiator

The Initiator Initialization routine (module IEFSD160) is entered under three circumstances:

- It is entered when the Initiator Control routine (module IEEVICTL) issues an XCTL macro instruction to use the initiator as a subroutine.
- It is entered when the Initiator Interface Control routine (module IEFIIC) issues an XCTL macro instruction after receiving control from the System Task Control routine in response to a START command that specifies an initiator.
- It is entered when the Initiator Job Selection routine issues an XCTL macro instruction in response to a MODIFY command. The processing performed in this case is described in the section "Processing the START and MODIFY Commands."

When the Initiator Initialization routine is entered under the first circumstance, the Initiator subroutine is being invoked to initiate a system task, a job that was started via a START command, or a TSO task entered from a TSO terminal. (For a description of the TSO Control Program, see the TSO Control Program PLM.) The Initialization routine is passed the JCT for the task and, therefore, does not dequeue a job from the input queue, process the STOP or MODIFY commands, nor build an ECB list to dequeue another job.

When the Initiator Initialization routine is entered under the second circumstance, the Initiator subroutine is being invoked to initiate a job step task that is not specified in a START command. In this case, the initiator must select the job from the input queue.

The routine is passed an initiator entrance list (IEL), an initiator options list, and an initiator exit list. The routine determines the processing to be performed from the information in the lists. The routine performs a syntax check on the class names, force priorities, and limit priority in the START command. If no specification was made in the command, the routine performs the syntax check on the PARM field of the Initiator Procedure EXEC statement. If it finds a syntax error, the routine returns control to the System Task Control routine, and the Initiating Task is terminated. If no error is found, the routine proceeds with the initialization:

- It obtains main storage from subpool 253 for the linkage control table (LCT), for a two-level register save area, and for queue manager parameter areas (QMPAs) for the message class and input queues.
- It places a pointer to the Track Stacking Parameter List into each QMPA.
- It copies the number-of-buffers specification from the Master Scheduler Resident Data Area into the Track Stacking Parameter Area.
- It initializes the LCT with the QMPA pointers, UCB List pointer, the initiator options from the initiator options list, the address of the initiator exit list, the addresses of the JCT and the CSCB if the initiator is being used as a subroutine to process a system task, and the return address to the System Task Control routine.

The following processing takes place only when the Initiator subroutine is initiating jobs from the input queues:

- It inputs the force priorities, limit priority, and the priority at which the initiator was attached to initiate a job step task, in the LCT.
- It obtains main storage from subpool 253 for the Initiator ECB List (see Figure 14).
- It constructs the ECB List, setting the post bit on in each ECB.
- It scans the chain of Group Control Blocks. If no GCB in the chain contains the initiator procedure name, the routine constructs and initializes a GCB and adds it to the chain. If a GCB exists for the group, the routine increments the member count by one.

Finally, the Initiator Initialization routine places a pointer to the ECB List in the LCT, places a pointer to the LCT in General Register 1, and passes control to the Job Selection routine.

Selecting Jobs

The process of job selection takes place only if the initiator receives control to initiate a job step task that is not specified in a START command.

The process of job selection includes the following functions:

- Looking for work by scanning the initiator's ECB list for an indication that a queue entry is available.
- Waiting for work, if there are no queue entries available.
- Dequeuing a job from the first queue on the initiator's list that contains an entry.
- Determining whether direct system output (DSO) processing has been initiated for any classes in which the job has data sets, assigning direct system output control blocks (DSOCBs) as required, and constructing a QMPA for the system output messages that pertain to the job.

Most of these functions are performed in the Job Selection routine (module IEFSD161). The wait for work takes place in the Initiator Wait routine (module IEFSD105), which resides in the link pack area.

LOOKING FOR WORK

Whenever it is entered, the Job Selection routine determines whether an internal stop has been issued, indicating that a started task or TSO terminal task has been completely processed. If so, the routine frees storage and returns control to the System Task Control routine. If no internal stop has been issued, the routine determines whether a STOP or MODIFY command has been issued by testing the stop bit in the ECB pointed to by the communication parameter area (CPA) -- see Figure 13, by testing the verb code in the command input buffer (CIB), and by checking for the presence of data in the command input buffer. If no MODIFY or STOP command has been issued, the routine determines whether it must dequeue a job from the input queue. If a JCT has been passed to the initiator, there is no need for the routine to dequeue a job for processing from the input queue. Otherwise, the Job Selection routine must dequeue a job from the queue. It scans the initiator's ECB list (see Figure 14), looking for an ECB that has been posted. When it finds an ECB with the post bit on, the Job Selection routine builds a QMPA for the corresponding queue, using the area obtained in the Initialization routine. When the QMPA has been constructed, the Job Selection routine uses a BALR instruction to pass control to the Queue Management Dequeue routine. The Dequeue routine issues an ENQ macro instruction and reads the QCR into main storage to determine whether there are any jobs in the queue.

ECB Address	4
CIB Address	4

Figure 13. Communication Parameter Area (CPA)

Note: The ECB address is a pointer to the communications ECB that is posted by SVC 34 when a STOP or MODIFY command is issued. The CIB address is a pointer to the command input buffer that contains the command code and parameters from the STOP and MODIFY commands.

If there are no jobs in the queue, the Dequeue routine issues the DEQ macro instruction and adds the appropriate ECB in the initiator ECB list to the no-work chain. The Dequeue routine then returns control to the Job Selection routine, which continues its scan of the ECBs in the list, looking for a posted ECB.

WAITING FOR WORK

If there are no jobs in any of the input queues in the initiator's set, the Job Selection routine uses the WTO macro instruction to inform the operator that the initiator is waiting for work, then uses the XCTL macro instruction to pass control to the Initiator Wait routine.

The Initiator Wait routine (module IEFSD105) is always loaded into the link pack area. When it is entered, the routine releases the initiator's region and issues a WAIT macro instruction specifying the initiator's ECB list. The first ECB specified in the list is posted by the Command Scheduling routine (SVC 34) as a result of a STOP or MODIFY command that applies to the initiator. The other ECBs specified in the list are in no-work chain elements; they are posted by the Queue Management Enqueue routine when a job is enqueued in the corresponding queue. When any of the ECBs is posted, the Initiator Wait routine obtains a new region of main storage for the initiator, and passes control to the Job Selection routine.

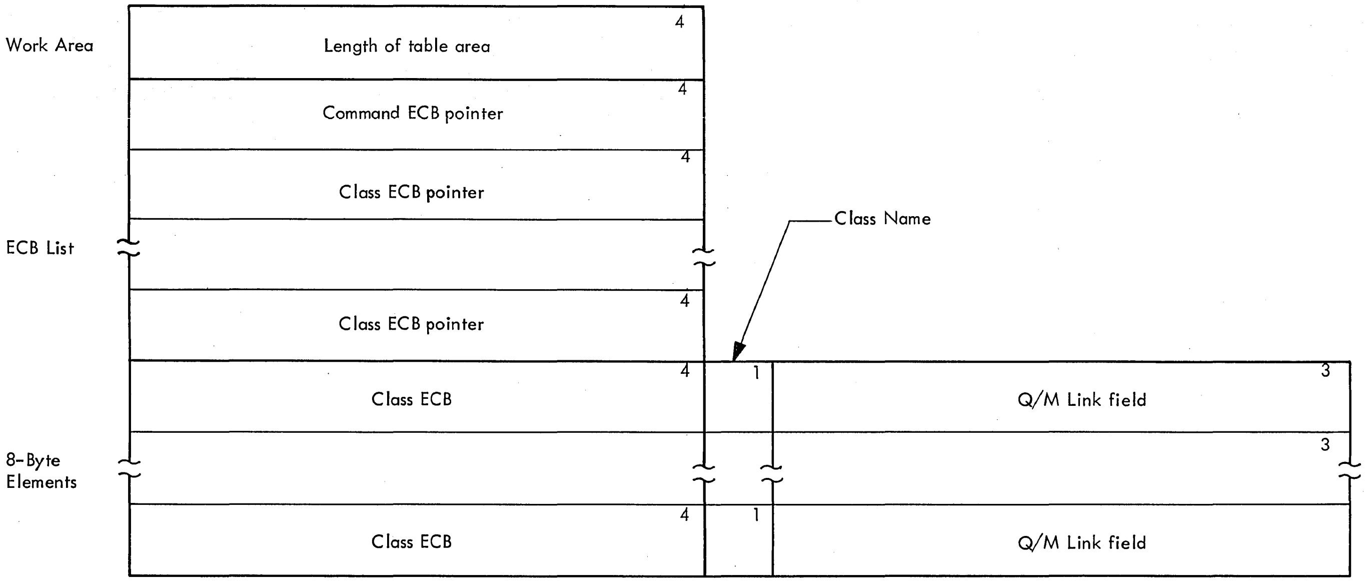


Figure 14. Initiator ECB List

DEQUEUING THE JOB

If there are jobs in the input queue, the Dequeue routine dequeues the highest priority job. When it has updated the input QCR, the routine writes the QCR back into the queue data set, then creates a CSCB for the job. It issues SVC 34 to have the Command Scheduling routine place the CSCB in the chain, then issues the DEQ macro instruction to permit access to the QCRs by routines performing other tasks.[1]

Before returning control to the Job Selection routine, the Dequeue routine reads the JCT from the job's queue entry into main storage. When it regains control, the Job Selection routine sets the delete bit in the CSCB on to indicate that this CSCB should be deleted from the CSCB chain and its main storage released when the job is completed. It then stores the address of the CSCB for the job in the LCT. Then, as is the case for all tasks, it obtains main storage in subpool 255 for the write-to-programmer control block (WTPCB) and storage in subpool 253 for the job step control block (JSCB). It places the addresses of the CSCB and the WTPCB in the JSCB, and the address of the JSCB in the LCT.

If more jobs are to be dequeued following completion of this job, the routine places the force priority and the limit priority in the LCT. Next the routine tests the job-canceled bit in the QMPA; if the bit is on, the routine sets the job-fail bit in the JCT on, sets the job termination bit in the LCT on, and reinitializes the address of the data set enqueue table in the JCT to zero so that data set integrity processing will not take place for this task.

The routine updates the track stacking information in the QMPA from the information in either the LCT or the initiator options list. It places the protect key from the TCB in the JCT and then extracts (from the JCT) the TTR of the SCT for the first step to be run. The Queue Management Read/Write routine reads the SCT into main storage.

SYSTEM OUTPUT PROCESSING

The Job Selection routine uses the information in the SCD to build a queue manager parameter area (in the second of the two areas obtained by the Initialization routine) for the system output message class. This parameter area is used when the I/O Device Allocation routine and the Termination routine add system message blocks to the job's message class queue entry, and when the Termination routine enqueues the job's output queue entries.

----

[1] When the DEQ macro instruction has been issued, the job has been dequeued. At that point, a CANCEL command applying to the job becomes an existing-task command instead of a task-creating command (see the Command Processing section of this publication).

The programs processing a job can write system output directly on an output device only if a direct system output block (DSOCB) for the job class has been assigned to the job. If DSO processing has been initiated, the Job Selection routine issues an ENQ macro instruction, in the share mode, specifying the chain of DSOCBs. It then uses the system output class directory (SCD) to determine whether there are any DSOCBs that can be assigned to the job. A DSOCB can be assigned if the following conditions are met:

- The DSOCB is available (not assigned to another job, and no commands pending).
- The DSOCB specified a system output class (including the message class) used in the job.
- No other DSOCB for that class has been assigned to the job.

If the conditions are met, the Job Selection routine makes the assignment by storing the initiator's protection key in the DSOCB's protection key field. The routine continues assigning DSOCBs to the job for other unassigned output classes until no more can be assigned, then sets the DSO indicator bit on in the JCT.

DATA SET INTEGRITY

If the address of the data set enqueue (DSENQ) table is not zero, or if the bit in the CSCB indicating "bypass data set integrity processing" is not on, the Job Selection routine reads the DSENQ table into main storage. (If the CSCB bit is on, or if the job was previously failed, the Job Selection routine places zeros in the DSENQ table address.) The DSENQ table, which may occupy several 176-byte records, contains the name of each non-temporary data set required by the job, and indicates whether the data set may be shared, or whether exclusive use of it is required. The routine obtains main storage (in subpool 255), then builds an ENQ macro instruction parameter list, specifying each name in the DSENQ table as well as its attribute (exclusive or share). Since duplicate names may exist in different records of the DSENQ table, these duplicates are eliminated; if the attribute of a duplicate name differs from the attribute of the original, the more restrictive attribute is assigned.

When the parameter list is completed, it is written back into the queue data set (in the space occupied by the DSENQ table), and a pointer to the list in subpool 255 is placed in the LCT. The ENQ SVC (SVC 56) is issued in the Replace Region routine (only on the first step of a job); the DEQ is issued in the Job Delete routine, when the job has been terminated.

Since the ENQ SVC is issued before a region is obtained for the first step of the job, no region will be obtained while any data set required for the job is under the exclusive control of another job. The ENQ SVC is issued with the USE option, so that a list of names of the available data sets is immediately returned to the initiator. Appropriate operator messages are then issued. Control finally passes to the Region Size Determination routine (module IEFSD101).

Executing the Stop and Modify Commands

When a STOP or MODIFY command specifies a group of initiators, the communications ECB of each member of the group is posted. In the case of a MODIFY command, the list of input queues to be associated with each initiator is stored in its command input buffer (CIB) and the member count field of the GCB is stored in the modify count field. In the case of a STOP command, the stop count field is incremented by one. The MODIFY CIB is pointed to by the CPA and the CSCB. The STOP CIB is pointed to by the GCB if one exists; otherwise, it is pointed to by the CPA and the CSCB.

If the initiator is waiting for work and one of the ECBs in an initiator's ECB list is posted, the Initiator Wait routine obtains a new region for the initiator and passes control to the Job Selection routine. If the intiator is processing a job, processing continues until the Job Selection routine (module IEFSD161) receives control to dequeue another job. In either case, upon entry the Job Selection routine tests the communication ECB; if that ECB has been posted, a command has been issued, and the routine uses the ENQ macro instruction to prevent multiple accesses to the GCB. The routine then tests the stop count field in the GCB. If the field is not zero, the initiator is to stop; if the field is zero, the Initiator subroutine determines if a CIB is present in the chain pointed to by the communications parameter area. If a CIB is present, the initiator checks it for a STOP command. If a STOP command is to be processed, the initiator processes it. If the command is not STOP, the initiator checks the command input buffer for a MODIFY command. If the stop count is zero, and no MODIFY command has been issued, the initiator continues processing.

THE STOP COMMAND

If the initiator is to stop, the routine tests the command input buffer to determine whether a MODIFY command has also been issued. If so, it is not executed, but the Job Selection routine reduces the modify count in the GCB by one. It also reduces the member count by one, and then tests it. If the member count is zero, there are no other members in the group and the Job Selection routine deletes the GCB from the chain and frees the main storage it occupies. In any case, the routine issues the DEQ macro instruction for the GCB chain, frees the main storage occupied by the LCT and the ECB list, and returns control to the System Task Control routine.

THE MODIFY COMMAND

If the stop count is zero, the initiator is to continue processing. The Job Selection routine issues the DEQ macro instruction to permit routines performing other tasks to access the GCB, and then inspects the command input buffer. Unless the buffer is zero, a MODIFY command is to be executed, and the Job Selection routine uses the XCTL macro instruction to pass control to the Initiator Initialization routine.

The Initialization routine (Module IEFSD160) performs a syntax check of the class names, force priorities, and limit priority specified in the MODIFY command (now stored in the CIB pointed to by the CPA or CSCB). If the routine encounters a syntax error, it issues an error message to the operator and passes control to the Job Selection routine without executing the command. If no errors are found, the Initialization routine constructs a new ECB list, then passes control to the Job Selection routine.

Region Management

Modules of the initiator operate in various regions of main storage. When the Initiator Initialization routine is first entered, it operates in a region obtained by or for the System Task Control routine. Subsequently, initiator modules operating in the link pack area release and obtain main storage regions as required.

DETERMINING REGION CHARACTERISTICS

Whenever the initiator requests a region of main storage, it determines the characteristics (size, address, and hierarchy) required for the region. In making this determination, the initiator uses the following factors:

- Minimum initiator region size, which is specified by the user when the system is generated (see the Storage Estimates SRL, GC28-6551) and stored in the BAMINPAR field of the master scheduler resident data area. The minimum initiator region must be large enough to contain the largest I/O Device Allocation load module (and associated tables and work areas). The user may change the minimum initiator region size at IPL time when specifying system parameters.

- Minimum job step region size, which is calculated by the Nucleus Initialization Program (see the IPL/NIP PLM) if load module IEFSD061 is in the link pack area. Load module IEFSD061 contains the Termination, Job Selection, and Region Size Determination routines. The minimum job step region must be large enough to contain the tables and work areas associated with termination and job selection functions. The value is stored in the BAMIPAR2 field of the master scheduler resident data area and is equal to the minimum initiator region size less the size of load module IEFSD061.

A minimum size region is required for the initiator to attach a subtask and for termination upon return from the subtask. If load module IEFSD061 is loaded in the link pack area, the region is equal in size to the minimum job step region size. Otherwise, it is equal in size to the minimum initiator region size. The minimum size region may be located in either hierarchy zero or hierarchy one.

- Hierarchy 0 and hierarchy 1 region sizes, which may be specified in the EXEC statement, are stored in the step control table (SCT). These values specify the size (in each hierarchy) of the region required for the execution of the job step.

- Hierarchy 0 and hierarchy 1 region addresses, which are stored in the SCT when a job step is to be restarted from a checkpoint. These are the beginning addresses in each hierarchy of the region originally obtained for the job step.

RELEASING AND OBTAINING REGIONS

The initiator releases the current region of main storage, and obtains a new region, under the following circumstances:

- When the initiator must wait for work.

- Each time the initiator selects a step.

- Before the initiator attaches a job step, if load module IEFSD061 is in the link pack area and the requested region size is smaller than the minimum initiator size region.

- Before the initiator attaches a system task, if the requested region size is smaller than the minimum initiator size region.

- Upon return from processing a system task, if the requested region size for the system task is smaller than the minimum job step region size (necessary for termination).

When the Initiator subroutine is entered, the current region being used is large enough to accommodate the first load of the initiator. This region is used by the initiator until it begins processing a starting system task, selects a job and step, or determines that it must wait for work to become available.

If the initiator must wait for work, control is passed to the Initiator Wait routine (module IEFSD105) in the link pack area. The Wait routine releases the current region being used by the initiator and then issues a WAIT macro instruction.

When the post occurs, the Initiator Wait routine regains control, builds a parameter list for GETMAIN, and issues a conditional GETPART macro instruction for a minimum size region in hierarchy zero. If sufficient main storage is unavailable in hierarchy zero, the routine issues a conditional GETPART macro instruction for a minimum size region in hierarchy one. If main storage is again unavailable, the routine issues an unconditional request for a minimum size region in hierarchy zero.

When the region is obtained, the routine passes control to the Job Selection routine (module IEFSD161) to select a job and a step.

Each time the initiator selects a step, it frees the current region and obtains a new one in which to perform the I/O device allocation function.

The FREEPART and GETPART macro instructions are actually issued in the Free/Get Region (module IEFSD102), which is in the link pack area. But the processing required before the macro instructions are issued is performed in the Region Size Determination routine (module IEFSD101), which is a part of load module IEFSD061.

When the Region Size Determination routine is entered, it branches to the User Exit Initialization routine (module IEFSMFIE). If SMF is supported, this routine initializes and updates the timing control table (TCT), updates the job log portion of the job management record (JMR) and passes control to the user's Job Initiation Exit routine (module IEFUJI) or Step Initiation Exit routine (module IEFUSI). If SMF is not supported, the User Exit Initialization routine returns control to the Region Size Determination routine.

When it is entered, the User Exit Initialization routine determines whether a TCT already exists for this job. If not, it obtains main storage from subpool 253 for the TCT and for the first 40 bytes of the JMR. The routine initializes the TCT, then uses the Queue Management Read/Write routine to bring the JMR into main storage. It copies the first 40 bytes of the JMR into the area reserved for it, then issues the TIME BIN macro instruction and stores the job initiation start time and date in the JMR. If user exits were specified, the routine brings the job account control table (ACT) into main storage, then passes control to the user's Job Initiation Exit routine (module IEFUJI).

If the TCT already exists, the JMR is already in main storage. The User Exit Initialization routine issues the TIME BIN macro instruction, and stores the step start time and date in the JMR. If user exits are specified, the routine brings the step ACT into main storage, then passes control to the user's Step Initiation Exit routine (module IEFUSI).

On the return from the user's exit routine, the User Exit Initialization routine inspects the return code. If the return code specifies that the job is to be canceled, the routine sets the job-failed bit in the JCT.

If the step being initiated is the first step of the job, and if the SMCA options field indicates that SMF data set information is required, the routine tests the SMCA option field to determine whether SMF exits are specified.

If SMF exits are not specified, the routine uses the Queue Management Read routine to bring the job ACT into main storage (if SMF exits are specified, the table is already in main storage.) The routine constructs a type 20 SMF record and issues SVC 83 to have the record placed in the SMF data set.

The User Exit Initialization routine returns control to the Region Size Determination routine (module IEFSD101). This routine determines whether track stacking[1] is being used.

If so, and if the job step is not the first step of a TSO job, the routine uses the Stack Purge routine to write out any updated work queue records. Otherwise, and on return from the Stack Purge routine, the Region Size Determination routine scans the program properties table (module IEFSDPPT) to determine whether the program to be executed is a privileged program, i.e. it is allowed to receive the region size specified for the step and/or is noncancellable (except during allocation).

Note: The program properties table (IEFSDPPT) is a table for job step tasks like the linkage table (IEEVLNKT -- see Figure 34) for system tasks. IEFSDPPT contains the names of the job step tasks that are allowed privileged execution. The table consists of two parts: the first part contains the 8-character names, padded with blanks if necessary, of the job step tasks that will receive the region size requested; the second part of the table contains the names of the job step tasks that are noncancellable.

If the Region Size Determination routine finds that the job step task to be executed is allowed to receive the region size requested, it turns off the minimum

[1] See the "Input/Output Operations" portion of the section "The Work Queues" in this publication.

initiator region size bit in the LCT. If the job step task is non-cancellable, the routine turns off the cancellable bit in the CSCB and turns on the non-cancellable bit in the LCT. [Note: this task is cancellable only during allocation.]

The Region Size Determination routine then obtains 44 bytes of main storage (from subpool 253) for the GETPART work table (GWT).

The GWT (shown in Figure 15) is a work area for the Region Size Determination routine and for the Free/Get region routine. It is also used as the parameter list for the GETPART macro instruction. The Region Size Determination routine initializes the GWT as follows:

- It sets the region size and region address pointers.

- It multiplies the SCT-specified region size values by 1024, and stores the results in the low-order three bytes of the region size fields.

- It stores the end-of-list code in the high-order byte of the appropriate region size field, and it stores a zero in the high-order byte of the hierarchy 0 region size field (if necessary).

- It stores the region address specifications (if any) in the low-order three bytes of the region address fields, and it stores the appropriate hierarchy code in the high-order byte of each field.

- It sets the request to X'80', indicating an unconditional request, and it sets the operation code to X'F7', indicating a "get" region (as opposed to a "replace" region) request.

- It stores the SCT-specified hierarchy 0 region address (if any) in the original address specification field.

- It stores the constant X'F7' in the high-order byte of the BAMINPAR value field, and it stores the allocation region size value (from the BAMINPAR field of the master scheduler resident data area) in the low-order three bytes of the field.

- It stores the region size for the subtask, if the Initiator Attach routine (module IEFSD263) is to replace the region.

- It stores the region size required for termination processing of the subtask if a new region is to be obtained.

Offset				
Hex	Dec			
0	0	Region Size Pointer 4		
4	4	Region Address Pointer 4		
8	8	Request Code 1	Operation Code 1	Reserved 2
C	12	Hierarchy 0 Code 1	Region Address Request (or Zero) 3	
10	16	Hierarchy 1 Code 1	Region Address Request (or Zero) 3	
14	20	End-of-List Code or Zero 1	Hierarchy 0 Region Size Request 3	
18	24	End-of-List Code 1	Hierarchy 1 Region Size Request (or Zero) 3	
1C	28	Reserved 1	Region Size for Subtask (or Zero) 3	
20	32	Reserved 1	Region Size for Termination (or Zero) 3	
24	36	Reserved 1	Original Region Address Specification (or Zero) 3	
28	40	X 'F7' 1	BAMINPAR Value 3	

Figure 15. GETPART Work Table (GWT)

Before performing allocation for a subtask, the initiator must obtain a minimum initiator size region in either hierarchy 0 or hierarchy 1. The Replace Region Interface routine (module IEFSD101) determines which hierarchy will contain the minimum size region.

If a hierarchy is specified in the START command, it is passed to the initiator as an option. The region sizes requested for each hierarchy in the EXEC statement are added, and the total region is obtained in the hierarchy specified in the START command.

If no hierarchy is specified in the START command, the routine determines the hierarchy from the EXEC statement. The routine makes the checks shown in Figure 16 and decides the appropriate region size and hierarchy in which it should be allocated (shown in the last two lines of the Region Allocation Decision Table).

R0 = 0 (or not specified)	X	X	X	X	X	X	X	X	X	X	X	X	X																		
R1 = 0 (or not specified)	X	X	X	X										X	X			X		X		X		X		X		X		X	
R0 < MJR														X	X	X	X	X	X												
MJR < R0 < MIR																				X	X	X	X	X	X						
R0 > MIR																										X	X	X	X	X	X
R1 < MJR					X	X	X																								
MJR < R1 < MIR								X	X	X																					
R1 > MIR											X	X	X																		
R1 > 0																X	X		X		X		X		X		X		X		X
Job step task being initiated	X	X			X	X		X	X		X	X		X	X	X	X			X	X	X	X			X	X	X	X		
System task being initiated			X	X			X			X			X					X	X					X	X					X	X
IEFSD061 in link pack area		X	X			X	*		X	*		X	*		X		X	*	*		X	X		*	*			X	X	*	*
Region size allocated in H0	MIR	MIR	MJR	MIR	0	0	0	0	0	0	0	0	0	MIR	MJR	MIR	MJR	R0	R0	MIR	R0	R0	MIR	R0	R0	MJR	MIR	R0	R0	R0	R0
Region size allocated in H1	0	0	0	0	MIR	MJR	R1	MIR	R1	R1	R1	R1	R1	0	0	R1	R1	0	R1	0	R1	0	R1	0	R1	0	R1	0	R1	0	R1

Where: R0 = Region size requested in hierarchy 0
R1 = Region size requested in hierarchy 1
MJR = Minimum job step region size
MIR = Minimum initiator region size

H0 = Hierarchy 0
H1 = Hierarchy 1
* = IEFSD061 may or may not be in the link pack area

Figure 16. Region Allocation Decision Table

If the step is being restarted from a checkpoint, region addresses are also specified in the SCT. If load module IEFSD061 is in the link pack area, and the requested region size is less than the minimum size region, the address specified in the SCT must be recalculated to ensure that a minimum initiator size region can be obtained.

The address is calculated using the following formula:

$$A_1 = A - (BAMINPAR - S)$$

where

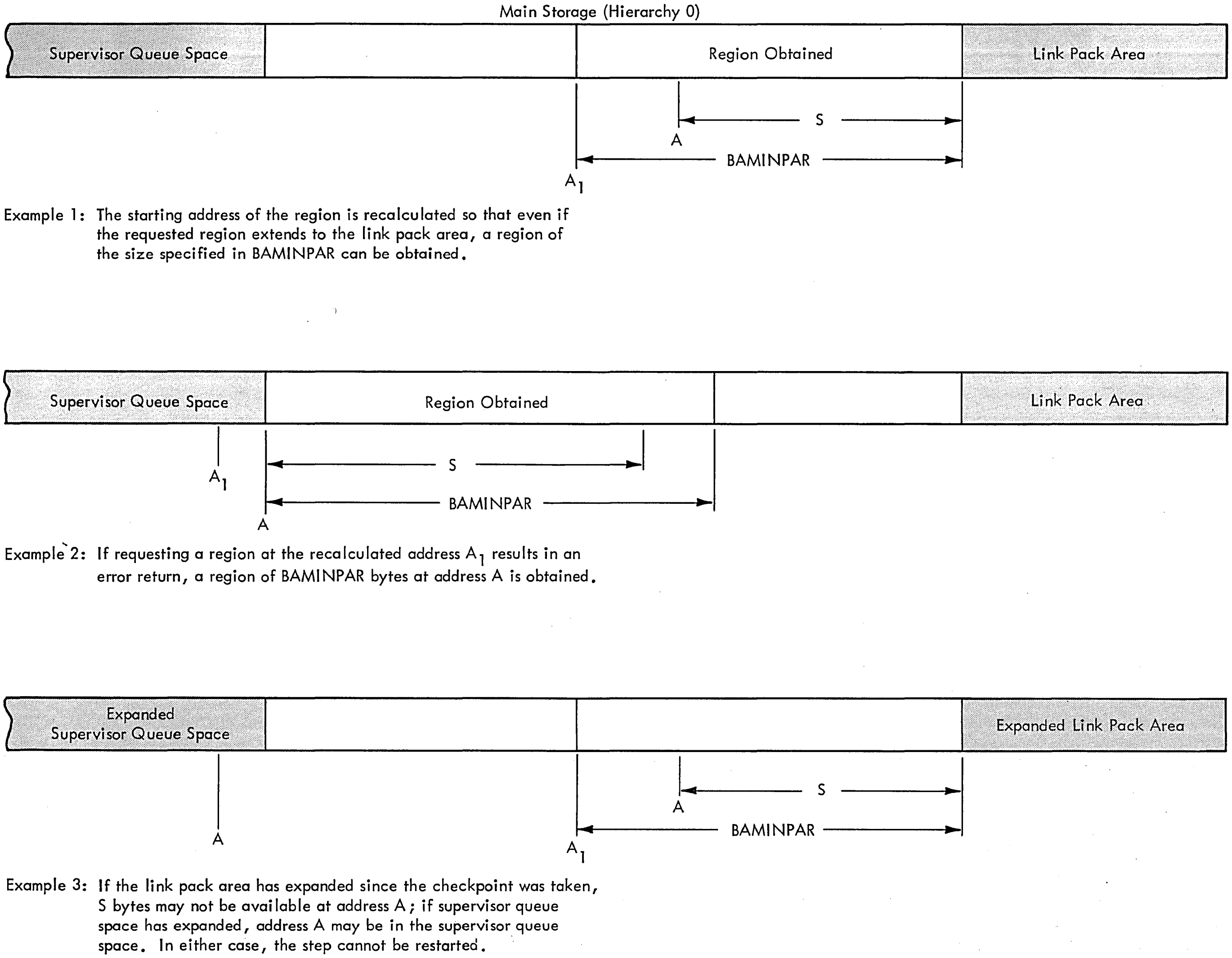
A_1 = recalculated hierarchy 0 or hierarchy 1 address
A = the hierarchy region address specified in the SCT
BAMINPAR = the minimum initiator region size
S = the SCT specified region size

If load module IEFSD061 is in the link pack area and the region size requested is less than the minimum initiator region size the Region Size Determination routine determines the region size required for the subtask by choosing the greater of the requested size or the minimum job step region size. The routine places this value in the GWT for later use by the Allocation Interface routine.

System tasks are executed in the region size requested in the EXEC statement. If the requested size is smaller than the minimum initiator region size (or the minimum job step region size if load module IEFSD061 is in the link pack area), the size necessary for termination is placed in the GWT by the Region Size Determination routine. This value will be used by the Initiator Attach routine (module IEFSD263) in later processing.

Next the Region Size Determination routine stores the address of the GWT in the LCTPARM3 field of the LCT and passes control to the Free/Get region routine (module IEFSD102), which is in the link pack area. Size routine passes the region size to the Replace Region routine.

The Free/Get Region routine (module IEFSD102) resides in the link pack area. When it is entered, it frees subpool 0 and the old region, then inspects the LCTPARM1 field in the LCT to determine whether an ENQ macro instruction need be issued. If the field is zero, the job uses only temporary data sets, or else the current step is not the first step of the job, and no ENQ need be issued. If the field is not zero, the routine issues a conditional ENQ (RET=USE) specifying data set names. If the routine is unable to obtain all of the data sets specified in the ENQ, it issues a DEQ (RET=HAVE) to free those that are controlled and writes messages to the operator indicating the names of data sets that are not available to the job. The routine then issues a WTOR macro instruction giving the operator the options of retrying the ENQ with RET=USE, cancelling (failing) the job, or waiting for the data sets to become available (RET=NONE). The routine then follows the instructions of the operator's reply.



Example 1: The starting address of the region is recalculated so that even if the requested region extends to the link pack area, a region of the size specified in BAMINPAR can be obtained.

Example 2: If requesting a region at the recalculated address A_1 results in an error return, a region of BAMINPAR bytes at address A is obtained.

Example 3: If the link pack area has expanded since the checkpoint was taken, S bytes may not be available at address A; if supervisor queue space has expanded, address A may be in the supervisor queue space. In either case, the step cannot be restarted.

Figure 17. Result of Issuing the GETPART Macro Instruction

When the ENQ is completed (either successfully or unsuccessfully), the Free/Get Region routine frees the main storage occupied by the parameter list pointed to by LCTPARM1 and issues the GETPART macro instruction (execute form), using the GWT as the parameter list. Example 1 of Figure 17 shows the effect, under normal circumstances, of issuing the macro instruction.

If an error return occurs, it means either that the requested region is larger than the total amount of main storage available for region allocation, or that a region address was specified and the recalculated address is in supervisor queue space.

If all main storage available for allocation is not sufficient to fill the request, the Free/Get Region routine stores an error code (8) in the LCT field LCTPARM3, and issues a GETPART macro instruction in the register form to obtain a minimum job step size region.

If the recalculated region extends into supervisor queue space, the routine stores the SCT-specified hierarchy 0 region address in the hierarchy 0 region address field of the GWT, then reissues the macro instruction using the execute form. The effect of reissuing the GETPART macro instruction is shown in Example 2 of Figure 17.

If an error return now occurs, it means that the main storage environment has changed since the checkpoint was taken. Either the link pack area, or supervisor queue space (or both) has expanded. Example 3 in Figure 17 shows each situation:

- If the link pack area has expanded, there is not enough main storage between the specified address (A) and the link pack area to satisfy the request.
- If supervisor queue space has expanded, the specified address (A) is in supervisor queue space, and the request cannot be satisfied.

In either case, the step cannot be restarted; the Free/Get Region routine stores an error code (8) in the LCTPARM3 field of the LCT and issues a GETPART macro instruction (register form) to obtain a minimum initiator size region.

When it has obtained a region of main storage, the routine passes control to the Allocation Interface routine (module IEFSD162) which is loaded into the new region.

The Allocation Interface routine inspects the LCTPARM3 field of the LCT; if the field indicates that the region could not be obtained, the routine sets the job failed bit on in the JCT, then passes control to the I/O Device Allocation routine.

Before the initiator attaches a job step (if load module IEFSD061 is in the link pack area) or a system task, it determines whether the region is to be replaced. The Allocation Interface routine (module IEFSD162) checks the GWT to determine whether the region size requested for the subtask is smaller than the minimum initiator region size. If not, the routine releases the main storage used by the GWT, since no replacement is necessary, and passes control to the TIOT Storage routine.

However, if the requested region size is smaller than the minimum initiator region size, the routine checks the LCT to determine which hierarchy contains the minimum size region and places the region size needed for the subtask in the appropriate region size field in the GWT. It then changes the operation code in the GWT to X'F6' to indicate a "replace" region instead of a "get" region and finally passes control to the TIOT Storage routine.

The actual replacement of the region is performed in the Initiator Attach routine (module IEFSD263). This routine, operating in the link pack area, issues the GETPART macro instruction in the execute form, using the GWT as a parameter list. After the region has been obtained, the routine checks the termination region size field in the GWT to determine if a new region size is required for termination. (If the current task is a job step task, the field will contain zero and no region will be obtained. However, if the task is a system task and the size requested for it is smaller than the minimum initiator region size, a new region will be obtained for termination.) If a new region is not required, the routine releases the GWT and attaches the subtask. If a new region is required, the routine checks the LCT for the hierarchy containing the minimum size region and places the termination region size in the appropriate region field in the GWT. It then changes the operation code to X'F7' indicating a "get" region.

Upon return from processing a system task, if a new region is required for termination, the Initiator Attach routine obtains the region and releases the storage occupied by the GWT.

The I/O Device Allocation Interface

When a region has been obtained for the job step or system task, the Allocation Interface routine (module IEFSD162) is entered. It obtains main storage from subpool 255 for the task parameter list (specified in the EXEC statement PARM field).

If the high-order byte of the track stacking parameter list is not zero, it means that track stacking is specified. The routine obtains a 72-byte register save area and passes control to the Stack Initialization routine (see the I/O Operations portion of the section on The Work Queues in this publication). When the track stack has been initialized, the Allocation Interface routine builds the allocation parameter list and determines whether the task is a job step that is being restarted from a checkpoint by checking if the program name in the SCT is IEFRSTRT. If so, the Allocation Interface routine passes control via a LINK macro instruction to the DSO Environment Check routine (module IEFDSOCR).

The DSO Environment Check routine uses the Queue Management Read routine to bring the SIOTs for the restarting step into main storage. It compiles a list of system output classes and unit types for which DSO processing is required and compares the list to the available DSOCBs. If the appropriate DSOCBs are available, the DSO Environment Check routine returns control to the Allocation Interface routine with a return code of zero.

If additional DSOCBs are required, the DSO Environment Check routine uses the WTO macro instruction to inform the operator, frees the job's resources, uses the Job Suspension routine (module IEFSD168) to enqueue the job on the hold queue, and returns control to the Allocation Interface routine with a return code of eight. The Allocation Interface routine then passes control to the Job Selection routine (module IEFSD161) to select another job.

If the return code is zero, or if the task is not a step restarting from a checkpoint, initiation of the task continues. The Allocation Interface routine issues a LINK macro instruction to pass control to the I/O Device Allocation routine.

The I/O Device Allocation routine (see part 6 of this publication) is used as a subroutine. It checks EXEC statement condition codes, then analyzes the requests for I/O devices and data sets, and allocates devices to the job step or system task. It issues operator messages to have any required volumes mounted, and creates the TIOT. Finally, it returns control to the Allocation Interface routine, with a return code that indicates whether the allocation process was successful.

On the return, the Allocation Interface routine inspects the return code. If the return code is not zero, it means that the task could not be run, either because of EXEC statement condition codes, or because the device or data set requirements of the task could not be satisfied, and control is passed to the Task Delete routine. If the return code is zero, the task can be run, and preparations are made to attach it.

Attaching the Subtask

The Supervisor routines need certain information in order to control the execution of a task. This information is passed to the supervisor via the ATTACH and STIMER macro instructions. In addition, information must be saved for the use of the Termination routine, and the task's TIOT must be moved to protected storage.

If a job step is to be attached, the Allocation Interface routine determines if DSO processing is to be performed (DSO processing is bypassed if a system task is to be attached). First the routine determines if the DSOCB indicator bit in the JCT is on and the DSO work bit in the SCT is on. If so, the Allocation Interface routine issues a LINK macro instruction to pass control to the DSO Writer routine (module IEFDSOWR). This routine writes job separators and system messages for SYSOUT classes that are specified for direct system output processing in this step. It scans the DSOCB chain for active DSOCBs assigned to the job. If an active DSOCB has its job separator bit off, the routine writes the appropriate job separator in the DSO data set and turns the job separator bit on. If the message class bit is on, the routine writes the accumulated SMBs and stores the TTR of the next SMB in the JCT.

If the routine encounters job queue errors while writing the job separators and/or system messages, it returns control with a return code of 12, and the Allocation Interface routine causes abnormal termination of the job. If the DSO Writer routine encounters I/O errors, it returns control with a return code of four, and the Allocation Interface routine fails the job. Otherwise, if no errors are encountered, the DSO Writer routine returns control to the Allocation Interface with a return code of zero.

Next, in the case of both a job step and a system task, the Allocation Interface routine (module IEFSD162) passes control to the Table Breakup routine (module IEFSD514). This routine converts the TIOT into a series of 176-byte records, and uses the Queue Management Read/Write and Assign routines to obtain space for the records and add them to the task's input queue entry. When the entire TIOT has been written out, the routine returns control to the Allocation Interface routine, which uses the Queue Management Read/Write routine to write the JCT and SCT into the work queue data set. Then the routine takes a pre-invocation exit if one is specified in the initiator exit list. If a nonzero return code results from the exit, the task is to be failed; the routine posts the CANCEL ECB with the appropriate completion code in the CSCB.

If a job step task is being attached, the routine determines whether the current step is the first step of the job to be run.[1] If so, and if joblib or fetch data sets have been specified, the Allocation Interface routine obtains main storage from subpool 253 for any necessary DCBs. The steplib and fetch data set DCBs are created and opened as required for the task.

If a job step is being attached, the Allocation Interface routine (module IEFSD162) determines whether allowing the step to use the full amount of time specified for it would cause the job time limit to be exceeded: the routine calculates the amount of job time remaining by subtracting the job time used from the job time limit, then compares this figure to the step time limit. It places the smaller of the two figures in the timer work area.

For both a job step and a system task, the Allocation Interface routine obtains storage from subpool 253 and builds the ATTACH macro instruction parameter list (see Figure 18) and the initiator parameter list (see Figure 19). The routine then passes control to the Pre-Attach routine (module IEFSD103).

Pointer to Program Name	4				
Pointer to Fetch or Joblib DCB	4				
Pointer to ATTACH ECB	4				
Jobstep-Owned Subpool	4				
Shared Subpool	4				
Reserved	4				
Dispatching Priority (DPMOD)	2	Limit Priority (LPMOD)	1	Reserved	1
Blanks	8				

Figure 18. ATTACH Macro Instruction Parameter List

[1] If the job is being restarted, the first step of the job is not necessarily the first step to be run. In such cases, the joblib and fetch DCBs are constructed and opened before the first step of the job is started.

Upon entry the Pre-Attach routine checks the CSCB to determine whether a system task or a job step task is being started. If a system task is being started, the routine writes the LCT for the task on the job queue. Next it obtains storage in subpool 253 for a 16-byte parameter area to be used by the Attach routine (module IEFSD263), an 8-byte area for saving track stacking information from the LCT, and a 72-byte area for saving the QMPAs from the LCT. The routine uses the Table Breakup routine to convert the LCT to 176-byte records, which are added to the task's input queue entry. The write-to-programmer control block (WTPCB) is updated to point to the system output QMPA in the JSCB.

The Pre-Attach routine determines the dispatching priority at which the system task will be attached. If a DPRTY value is specified for the system task, the routine converts that value to a dispatching priority. If the dispatching priority is less than or equal to the limit value (255), the routine places it in the ATTACH macro instruction list. If the dispatching priority exceeds the limit value, the routine places the limit value in the ATTACH macro instruction list.

If no DPRTY exists, but a PRTY does, the routine converts the PRTY value to a dispatching priority. If the dispatching priority is less than or equal to the limit value, the routine places it in the ATTACH macro instruction list. If the dispatching priority exceeds the limit value, the routine places the limit value in the ATTACH macro instruction parameter list.

If no priority is specified, the routine uses a default of 251.

Pointer to Job Step Parameter List
Pointer to Fetchlib DCB
Pointer to ATTACH Parameter List
Pointer to LCT
Pointer to Initiator TCB
Pointer to TIOT List

Figure 19. Initiator Parameter List

If a job step task is being started, the Pre-Attach routine determines the priority at which the job step will be attached.

If a force priority is associated with the class from which the job is selected, and it is equal to or less than the limit value, it is put into the ATTACH macro instruction parameter list. If the force priority is greater than the limit value, the limit value is put into the ATTACH macro instruction parameter list.

If no force priority exists, and if a DPRTY value is specified for the job step, the Pre-Attach routine converts it to a dispatching priority in the same manner that it does for a system task (see above explanation). If no DPRTY value exists, the routine converts the PRTY value to a dispatching priority.

For all tasks, the routine initializes the ATTACH macro instruction parameter list, then passes control to the Attach routine (module IEFSD263).

The Attach routine (module IEFSD263) resides in the link-pack area. If SMF is in the system, the Attach routine first passes control to the TCTIOT Construction routine (module IEFSMFAT). This routine determines the lowest addresses allocated at the high end of the hierarchy 0 and 1 regions and the highest addresses allocated at the low end of the hierarchy 0 and 1 regions, then calculates the amount of unused storage. It stores these figures and the wait time limit in the TCT. If a background job is being processed, the routine places into the TCT the addresses of both the User Time Limit Exit routine (module IEFUTL) and the SYSOUT Limit routine (module IEFUSO). If a foreground job is being processed, the routine bypasses the placing of these addresses in the TCT. Finally, the routine obtains main storage from subpool 253 and constructs the timing control task input/output table (TCTIOT).

When it has constructed the TCTIOT, module IEFSMFAT issues a TIME macro instruction (SVC 11) to obtain a time stamp that indicates the time at which loading of the problem program began. After storing the time stamp in the TCT, the routine passes control back to the Attach routine (module IEFSD263). If the task is a TSO task, the Attach routine exits to TSO module IKJEFLM. Otherwise, the Attach routine moves the job step parameter list from subpool 253 to subpool zero, and frees the main storage obtained from subpool zero. The ATTACH macro instruction is then used to attach the task. If a job step is attached, the STIMER macro instruction is used to set the step time interval. The Attach routine then relinquishes control to the CPU by waiting for a POST macro instruction specifying either the cancel, attach, or dynamic allocation ECB. (The dynamic allocation ECB is posted by dynamic allocation routines to permit data set integrity processing.)

Terminating Subtasks

An executing task is terminated under one of the following circumstances

- A task requests normal or abnormal termination, or is abnormally terminated by the supervisor because of an error condition; the attach ECB is posted, and the Attach routine is given control.
- A CANCEL command is issued; the Command Scheduling routine posts the cancel ECB, and the Attach routine is given control.
- A job step fails to return control before its time interval has expired; an Asynchronous Exit routine, activated by the timer, posts the cancel ECB, and the Attach routine is given control.
- A job step exceeds the system output record limit specified in the DD statement OUTLIM parameter.

OUTLIM TERMINATION

If a job step exceeds the system output record limit specified in a DD statement OUTLIM parameter, an SMF asynchronous exit routine (module IEATLEXT; see the MVT Supervisor PLM) is given control. IEATLEXT constructs a parameter list containing the address of the JMR job name and time stamp fields and the address of the DCB. Using a branch and link instruction, IEATLEXT passes control to the SMF User Output Limit routine (module IEFUSO), which is located in load module IEFSD263 in the link pack area.

The user may supply a routine named IEFUSO that determines whether the job step is to be terminated. If the step is not to be terminated, the user-supplied routine must place the number of additional records to be permitted the step in register 1, and return control to IEATLEXT with a return code of 4 in register 15.

If the step is to be terminated, the user-supplied routine returns control to IEATLEXT with a return code of 0 in register 15. If the user does not replace the IBM-supplied version of IEFUSO, it will return control with a code of 0. On the return, IEATLEXT passes control to the Attach routine.

THE ATTACH AND ABNORMAL TERMINATION ROUTINES

In the case of an OUTLIM termination, the Attach routine (module IEFSD263) receives control from module IEATLEXT (see the above explanation of OUTLIM TERMINATION). In the case of a CANCEL or timer termination, the POST macro instruction issued as a result of the CANCEL command or timer expiration causes an SVC interruption to occur. Therefore, even though the task may be executing, it is interrupted. Since the initiator (which was in the wait state until the post occurred) has a slightly higher priority than the task, the Attach routine, operating under the initiator TCB, is given control first.

When the Attach routine receives control, it determines whether the timer was set before the task was attached. If so, the task is a job step, and the routine issues the TTIMER macro instruction (with the cancel option) and places the time remaining to the job step in the timer work area. Next, the routine inspects the cancel ECB; if it has been posted, the Attach routine prepares to pass control to the Abnormal Termination routine. Otherwise, the routine bypasses abnormal termination processing.

The Attach routine passes control to the Abnormal Termination routine when a task must be prevented from further use of system resources because of a CANCEL command, or because its time interval has expired, or because it has exceeded the number of records specified in a DD statement OUTLIM parameter. If the dispatching priority of the task to be cancelled is less than that of the initiator, the Attach routine issues a CHAP macro instruction to make the task's priority equal to that of the initiator. The Attach routine then builds a parameter list to pass to the Abnormal Termination routine via register 15. The first word of the parameter list is the address of the job step TCB. The second word is the completion code that was posted with the cancel ECB. The routine also sets the appropriate abnormal termination indicator in the CSCB and a bit in the task TCB to indicate that no STAE routines are to be executed. Then the routine issues SVC 34 to call the Command Scheduling routine, which branches to the Abnormal Termination routine.

When control returns to the Attach routine, it relinquishes control by issuing a WAIT macro instruction that specifies the attach ECB. After the initiator enters the wait state, the job step is abnormally terminated. The Attach routine gets control as a result of a POST macro instruction issued when the abnormal termination is complete.

If the terminating task is a TSO task, the Attach routine passes control to TSO module IKJEFLM. If the terminating task is a job step task, the Attach routine saves the step's TCB flags, the TCT address, the task completion code in the LCT, and the dispatching priority. If a system task is terminating, the routine saves the TCB flags, the TCT address, the task completion code, and the dispatching priority in the parameter area obtained by module IEFSD103. The routine releases subpool zero and issues a DETACH macro instruction to remove the TCB from the queue and to release the storage it occupied.

If a system task is terminating, the routine determines if the task has been using less than a minimum initiator size region (or a minimum job step size region if load module IEFSD061 is in the link pack area). If so, the routine obtains a new region for termination and passes control to the Post-Attach routine (module IEFSD104).

Upon entry, module IEFSD104 checks the parameter list that was built by the Attach routine to determine whether a system task or a job step task is terminating. If a system task is terminating, the routine obtains storage in subpool 253 and uses the Table Breakup routine to read the LCT from the job queue and place it in the area obtained in subpool 253. Next the routine moves the QMPAs and track stacking information saved in subpool 253 by the Pre-Attach routine (module IEFSD103) into the LCT. The routine then updates the WTPCB with the address of the system output QMPA. Next it releases the storage obtained in subpool 253 by module IEFSD103 for the parameter area, track stacking information, and QMPAs.

The Post-Attach routine takes a post-invocation exit if one is specified in the initiator exit list, passing the LCT. Then control passes to the Task Delete routine (module IEFSD164).

THE TERMINATION ROUTINE INTERFACE

The Task Delete routine (module IEFSD164) is entered from module IEFSD104 whenever a task terminates. Its primary function is to create an interface with the Termination routine (see part 6 of this publication), which is used as a subroutine to direct the disposition of any data sets used by the task, to execute the User's Accounting routine, to check JOB statement condition codes, and to release the I/O devices allocated to the task. It also uses the CHAP macro instruction to return the initiator's priority to its original level.

When a task terminates, the initiator parameter list, and the fetch, steplib, and joblib DCBs are in main storage areas previously obtained for them. Needed information is extracted from the initiator parameter list, the WTPCB for the task being terminated is placed in the JSCB pointed to by the active TCB, the fetch, steplib, and joblib DCBs are closed, the main storage areas are released, and the WTPCB pointer for the initiator is replaced in the JSCB pointed to by the initiator's active TCB.

The Termination routine needs the information in the JCT, the SCT, the WTPCB, and the job and step account control tables (ACTs). The Task Delete routine determines whether track stacking is specified. If so, it uses the Stack Initialization routine to build a stack, then uses the Queue Management Read routine to bring the JCT, SCT, and ACTs into main storage. It places the step time in the step ACT, then adds the step time to the job ACT, before writing the ACTs back into the queue data set. The routine creates a dummy TCB for termination and initializes it with information passed from the Attach routine. It then issues a branch and link instruction to pass control to the Termination routine.

When the Termination routine returns control, the Task Delete routine releases the dummy TCB.

If a job step is terminating and if the step is the last step of the job (and the job is not to be restarted), the Task Delete routine reinitializes the write-to-programmer control block (WTPCB) to zeroes and passes control to the Job Delete routine.

If the step is not the last step of the job (and the job is not to be restarted) the Task Delete routine reinitializes the WTPCB for use by the next step of the job, uses the Queue Management Read/Write routine to bring the SCT for the next step into main storage, and passes control to the Region Size routine.

If the job is to be restarted, the Task Delete routine reinitializes the WTPCB to zeros, sets the origin class ID field of the QMPA to indicate that the job is to be reenqueued in the hold queue (it will later be moved to an input queue), and passes control to the Job Suspension routine.

Unexecuted Steps

When a step cannot be run, the Allocation Interface routine also passes control to the Task Delete routine (module IEFSD164). Certain termination functions must be performed even though the step was never attached; system input data sets, for example, must be deleted. The Task Delete routine must therefore provide an interface for the Termination routine identical to the interface provided when a step has been executed.

Since no ATTACH macro instruction parameter list was created, the storage cannot be released. Since the SCT and JCT are already in main storage, they need not be read in again, and since no ATTACH macro instruction was issued, there is no job step TCB. Thus the Task Delete routine need only create a dummy TCB, and initialize it. The routine then uses the branch and link instruction to pass control to the Termination routine.

When the Task Delete routine regains control, it uses the FREEMAIN macro instruction to release the main storage occupied by the dummy TCB. If the step is the last step of the job, the routine passes control to the Job Delete routine.

If, however, the step is not the last step, the Task Delete routine updates the WTPCB for use by the next step, uses the Queue Management Read routine to bring the SCT for the next step into main storage, then passes control to the Region Size routine.

Job Suspension

When a job is to be restarted or when the necessary DSO resources are not available, the Job Suspension routine (module IEFSD168) is used to suspend processing of the job and to initiate the process of restarting. When it is entered, the routine uses the Queue Management Read/Write routine to update the SYSOUT class directory (SCD) in the QMPA. If any DSOCBs were allocated for the job, the routine uses the DSO Free Block routine (module IEFDSOFB) to free the DSOCBs. Then the Job Suspension routine issues a DEQ macro instruction specifying the names of data sets for which the Replace Region routine (module IEFSD102) issued an ENQ macro instruction. Next the routine reenqueues the JCT on the job queue, or on the hold queue for DSO, at a priority of 14 for restart. Finally, the routine frees the main storage occupied by the SCT and the JCT. If the job is to be restarted, the routine passes control to the Restart Activation routine (module IEFVSDRA).

The Restart Activation routine (module IEFVSDRA) constructs a START command containing the name of the job to start the Restart Reader. It issues the MGCR macro instruction to schedule execution of the command, then returns control to its caller, the Job Suspension routine.[1]

On the return, the Job Suspension routine frees the WTPCB and the JSCB for the job being suspended and passes control to the Job Selection routine. The Initiator subroutine than continues processing.

JOB TERMINATION

When the last step of the job terminates, the Task Delete routine passes control to the Job Delete routine (module IEFSD166). This routine uses the Queue Management Read routine to read the DSENQ table into main storage. It builds a DEQ macro instruction parameter list, and issues the DEQ macro instruction with the RET=HAVE option, making the data sets referred to by this job available to other jobs. It uses the Queue Management Delete routine to remove the job from the input queue, and then releases the main storage occupied by the JCT and the SCT. It uses SVC 34 to delete all CIBs on the CIB chain pointed to by the CSCB for the executed task. If the delete bit is on in the CSCB, the routine then issues SVC 34 to have the Command Scheduling routine delete the job's CSCB from the CSCB chain, and release the main storage it occupies. The main storage occupied by the JSCB and the WTPCB for the task is released by the Job Delete routine. Then, if only one job is to be processed (indicated by the options in the LCT), the routine sets the internal stop switch in the LCT on. For non-TSO tasks, module IEFSD166 issues a CHAP macro instruction with a priority change value of zero. This causes the current (or active) initiator to be placed below all other initiators of the same priority on the TCB queue. Thus all initiators of the same priority have an equal chance to be dispatched. Finally, the routine passes control to the Job Selection routine.

[1] The Restart Activation routine is also used in the System Restart routine (see the section "Common Elements" in this publication).

Part 4: Processing System Output

System output consists of data sets and messages from the operating system to the programmer. The messages, and pointers to the data sets, are placed in system output queues by the interpreter and initiator; they are retrieved by routines performing writing tasks, and written on devices designated by operator command for system output.

There are 36 classes of system output permitted in the operating system, and there is an output queue that corresponds to each class. The programmer defines a data set as being a system output data set by using the SYSOUT keyword in the DD statement; he specifies the class to which it belongs in the parameter associated with the SYSOUT keyword. For each system output data set, the interpreter assigns space in the appropriate output queue entry for a data set block (DSB), which points to the data set's JFCB.

The programmer may specify the class that is to contain operating system messages pertaining to his job by using the MSGCLASS keyword in the JOB statement. System message blocks (SMBs) containing interpreter, allocation, and termination messages are built, and placed in entries for the designated queue. If the programmer makes no message class specification, the system uses the class designated as the message class when the system is generated.

Each output queue entry describes the system output, of the corresponding class, for one job. If the entry corresponds to the message class, it contains SMBs, and may also contain DSBs, if there are system output data sets in that class; if the entry does not correspond to the message class, it contains only DSBs.

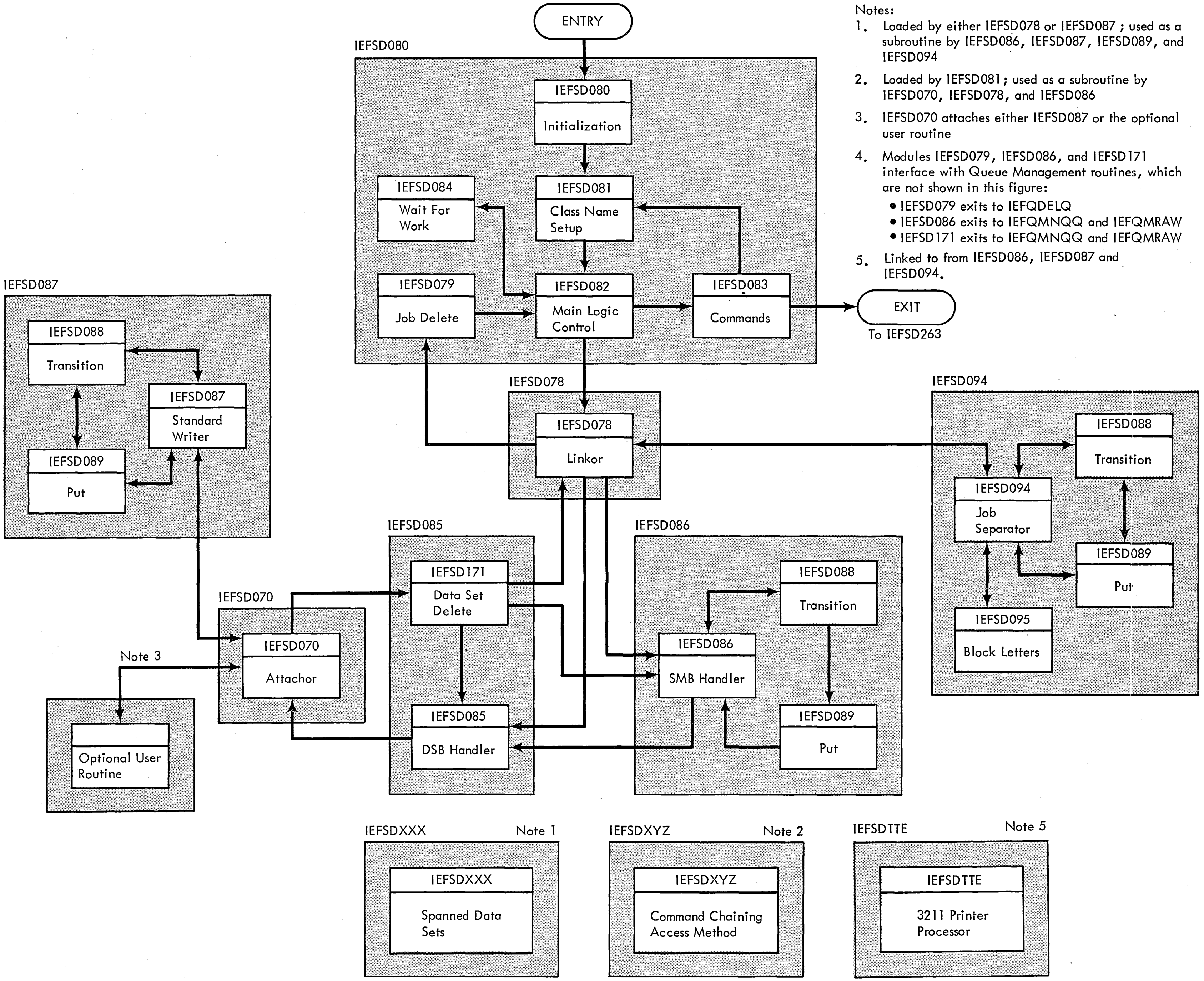
There are two ways in which system output is written on the appropriate output device:

- The operator may use START commands to initiate direct system output (DSO) processing for one or more classes.
- The operator may use START commands to establish writing tasks for system output writers.

If DSO processing is started, a job having system output in a DSO class writes data directly on the appropriate device at job and step initiation and termination. It writes system output in other classes on a direct access device for processing by system output writers after the job has completed execution.

The task of a system output writer (established as a result of a START command specifying a writer) is to dequeue output queue entries by class, and to write the messages and data sets on its output device. In its execution of a writing task, the system output writer (see Figure 20) performs the following functions:

- It initializes itself according to the parameters in the cataloged writer procedure and the START command, and reinitializes itself in response to MODIFY commands.
- It uses Queue Management routines to dequeue system output queue entries. If there are no entries in the queue that corresponds to the first class specified in the cataloged procedure or command, it tries the next. If there are no entries in any of its classes, the writer enters the wait state until a job terminates, and the Termination routine enqueues an entry in an output queue corresponding to one of its classes.
- It uses the GET/PUT Level Data Management routines to read records from system output data sets, and to write those records, as well as system output messages, on its output device. If a User routine has been specified to perform this function, the writer passes control to the User's routine.



Notes:

1. Loaded by either IEFSD078 or IEFSD087; used as a subroutine by IEFSD086, IEFSD087, IEFSD089, and IEFSD094
2. Loaded by IEFSD081; used as a subroutine by IEFSD070, IEFSD078, and IEFSD086
3. IEFSD070 attaches either IEFSD087 or the optional user routine
4. Modules IEFSD079, IEFSD086, and IEFSD171 interface with Queue Management routines, which are not shown in this figure:
   - IEFSD079 exits to IEFQDELQ
   - IEFSD086 exits to IEFQMNQQ and IEFQMRAW
   - IEFSD171 exits to IEFQMNQQ and IEFQMRAW
5. Linked to from IEFSD086, IEFSD087 and IEFSD094.

Figure 20. System Output Writer Overall Flow

Initializing and Processing Commands

The System Output Writer is entered at the Writer Initialization routine (module IEFSD080) via an ATTACH macro instruction issued in the Attach routine of the Initiator subroutine. The Initialization routine moves the DD name from the Writer's TIOT to the DCB for the output data set, then reads the JFCB into main storage.

If, upon inspecting the output UCB, the routine determines that the output device is a 3211 UCS rpinter, the UCS and FCB image-ids established at START WTR time are saved as default ids in the writer communication area--PARLIST. If inspection of the output UCB indicates that the output device is a 1403 printer with the UCS feature, the initialization routine saves the UCS image-id in the manner just indicated. These default ids are subsequently used by the SMB handler (IEFSD086), the Standard Writer Routine (IEFSD087), and the Job Separator (IEFSD094) if new image-ids are not defined and currently loaded image-ids are not default image-ids.

The routine inspects the JFCB to determine whether the writer is to use the standard QSAM Put Locate routine, or whether it is to use a special command chaining access method. It uses the special access method if all of the following conditions are met:

- The writer uses a printer or punch as the output device.
- The writer uses machine control characters.
- The writer does not use PCI.
- The writer uses four or more buffers for the output data set.

If all of the conditions are met, the Writer Initialization routine sets a switch, frees the main storage occupied by the JFCB, and passes control to the Class Name Setup routine.

The Class Name Setup routine (module IEFSD081) initializes the writer to process a specified set of system output classes. The routine is first entered from the Writer Initialization routine; it is subsequently entered from the Command routine to process the MODIFY command.

When the Class Name Setup routine is first entered, it determines whether the special access method is to be used. If so, the routine obtains main storage for, and constructs an IOB, an ECB, the required CCWs, the required buffers, and an EXCP DCB (which overlays the QSAM DCB).

Whether the special access method is to be used or not, the routine opens the output data set. If the output device is a UCS printer, the UCS/FCB buffers are loaded by OPEN to that environment specified in the writer's output DD card, the WTR PROC or the START WTR command. If the special access method is to be used, the routine uses the LOAD macro instruction to bring the Command Chaining Access Method routine into main storage.

Whenever the Class Name Setup routine is entered, it performs validity checks on the contents of the PARM field in the procedure EXEC statement (if the command is a START command), and on the command itself. Then, using the validated information, the routine creates and modifies the list of classes to be processed by the writer, stores the information used in translating carriage control characters, and sets up the pointers and ECBs used in communicating with other tasks. The table area created by the Class Name Setup routine is shown in Figure 21.

The first three words of the table area are used as a work area. The first word contains the length of the table area. The second word is used by the Main Logic routine to store the queue address of the first record of the current output queue entry, in case a permanent error makes it necessary to re-enqueue the entry when the writer is stopped. The third word of the work area contains three indicators in the high-order bits: The first bit (the 1442 Punch bit) is set on to indicate that the output device is a 1442 card read punch; the second bit (the punch type output bit) is set on when the output device is an intermediate device, and the records it holds will eventually be punched; the third bit (the printer or punch bit) is set on when the output device is a printer or punch. These indicators are used in translating carriage control characters from input data sets.

The ECB pointer list contains, as its first entry, the address of the communications ECB in the CSCB. This ECB is posted by the Command Scheduling routine whenever there is a command for the writer to process. The remainder of the entries in the ECB pointer list contain the addresses of the class ECBs, in the order in which the corresponding classes are to be processed by the writer.

The 8-byte elements associate a class ECB with a class. There is an element for each class specified in the START or MODIFY

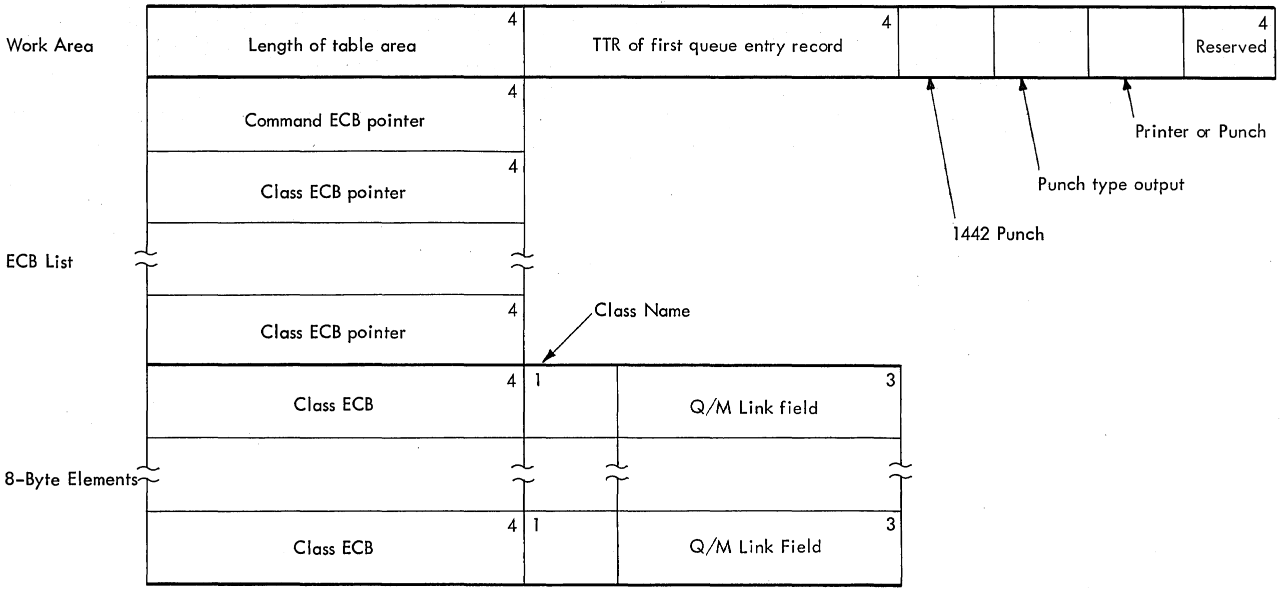


Figure 21. System Output Writer Table Area

command; the elements are arranged in the same order as specified in the command. The first word in each element contains the class ECB, which is initialized as posted, and is subsequently posted by queue management whenever an entry in that class is enqueued. The second word in the element contains, in the high-order byte, the class name. The low order three bytes contain a queue management linkage field. When a dequeue request is made that cannot be satisfied (because there are no entries in that queue), the Queue Management Dequeue routine adds the element to a chain of elements associated with writers that process the class.

When the table area has been constructed, control is passed to the Main Logic Control routine.

The Main Logic Control routine (module IEFSD082) is entered whenever the system output writer is ready to process an output queue entry or a command, and whenever a permanent I/O error is encountered. If a command is to be processed, or an I/O error has been encountered, the routine passes control to the Command routine. If a queue entry is available, the routine dequeues it, and passes control to a Linkor routine. If no work is available, the Main Logic Control routine passes control to the Wait routine.

The Main Logic Control routine first inspects the return code passed to it by a Queue Management or Writer routine; if the code indicates that a permanent I/O error has been encountered, the Main Logic Control routine passes control to the Command routine. If no I/O error was encountered, the Main Logic Control routine determines whether a command is to be processed by inspecting the command ECB; if the ECB has been posted, a command has been issued, and control is passed to the Command routine.

The Command routine (module IEFSD083) uses the Queue Management Unchain routine to unchain each class ECB. If the command is MODIFY, control is then passed to the Class Name Setup routine. If the command is STOP WTR, or if a permanent I/O error was encountered, the Command routine closes the output data set, frees the storage obtained for the table area, informs the operator, and returns control to the Initiator subroutine.

If there is no command to process, the Main Logic Control routine looks for a class ECB that has been posted. It examines the class ECBs (in the order in which the class names were specified); when it finds a class ECB in the posted state, it uses queue management to dequeue the highest priority entry in the corresponding queue. If the dequeue operation is successful, the Main Logic Control routine passes control to the Linkor routine (module IEFSD078).

If, however, the queue contains no entries (because other writers have processed all entries since the POST macro instruction was issued), the Main Logic Control routine looks for another posted class ECB.

If none of the ECBs are in the posted state, or if there are no available entries that correspond to posted ECBs, control is passed to the Wait routine (module IEFSD084), which issues a WAIT macro instruction specifying the ECB pointer list. The writer is thus placed in the wait state until either the command ECB or a class ECB is posted. When the Wait routine regains control, it passes control back to the Main Logic Control routine.

Processing a Queue Entry

An output queue entry is made up of DSBs (one for each data set specified by the job to be in that class); if the queue corresponds to the message class, the queue entry also includes the SMBs for the job. The records in the queue entry are linked together so that each record points to the next. When the Main Logic Control routine dequeues an entry, the first record is read into main storage, and the Main Logic Control routine passes control to the Linkor routine.

The Linkor routine (module IEFSD078) examines the output DCB, and if the output data set is to contain VS (variable-length spanned) records, the routine uses the LOAD macro instruction to bring the Spanned Data Sets routine (module IEFSDXXX) into main storage.

If a separator program has been specified to identify the output resulting from processing a queue entry, the Linkor routine passes control to it. The IBM-supplied Separator program consists of two modules:

- The Job Separator routine (module IEFSD094), which is the control routine. When it is entered, this routine initializes itself, and then examines the output UCB device type. If it is a 3211 printer, the Job Separator determines if the currently loaded UCS/FCB image-ids are default images and if data checks are blocked. If not, the Job Separator issues the LINK macro instruction to the 3211 Processor Module (IEFSDTTE) to load the writer default image-ids saved in the PARLIST parameter list by the Initialization routine (IEFSD080) and to block data checks during Job Separator processing. On return from the 3211 Processor Module the Job Separator routine uses the Transition routine (see the section on the service routines), and determines whether the output data set is to be printed or punched. It uses the Block Letters routine (the other module of the Separator program) to generate the necessary records, and it uses the Put routine to place them in the output data set. It uses the Transition routine again, then returns control to the Linkor.

- The Block Letters routine (module IEFSD095), which generates the records that make up the block letters and numbers. It is entered from the Job Separator routine, and when it has generated the requested record, it returns control to the Job Separator routine.

When the Separator program is entered (at the Job Separator routine) it executes the Transition routine and then determines whether the output data set is to be printed or punched.

If the output data set is to be printed, the Job Separator routine uses the Block Letters routine to generate the records necessary to print the job name and system output class name (in block letters) on each of the pages. The Job Separator routine generates the records necessary to overprint a line of asterisks on the fold preceeding each page, and on the fold following the last of the three pages.

If the output data set is to be punched, the Job Separator routine uses the Block Letters routine to generate the records necessary to punch the job name and system output class name in each of three cards.

In either case, the Job Separator routine uses the Put routine (module IEFSD089) to place the records in the output data set. When the last record has been passed to the Put routine, the Job Separator routine executes the Transition routine, then returns control to the Linkor routine (module IEFSD078).

The Separator routine determines whether the command chaining access method is being used. If so, it passes control to the Command Chaining Access Method routine (module IEFSDXYZ), which truncates the chain of CCWs and writes out any filled (but unwritten) buffers, then returns control to the Separator.

On the return from the Separator routine, the Linkor routine examines the entry and passes control to the appropriate processor, depending on whether the record is an SMB or a DSB. The processors read the subsequent records, and pass control to each other as the record type changes. When the entry has been processed, control is returned to the Linkor routine, which passes control to the Job Delete routine.

The Job Delete routine (module IEFSD079) uses queue management to delete the job's output queue entry from the queue. If an I/O error occurred during processing of the entry and SMF is supported, the routine stores an error indicator before issuing SVC 83 to have the SMF record (type 6) transferred to the SMF buffer. On the return, the Job Delete routine passes control to the Main Logic Control routine.

PROCESSING DATA SETS

The characteristics of a writer's output data set are fixed when the writer is started. The characteristics of its input data sets may vary widely, and are unknown until the writer is ready to process a data set. The writer must therefore dynamically initialize itself to process any one of a large variety of inputs, and it must be able to reformat input records and translate control characters to fit the characteristics of the output data set. If the output device is a 3211 UCS printer, the UCS/FCB buffers can be dynamically loaded prior to processing each data set.

The initialization is performed by the DSB Handler routine each time a data set is to be processed; the reformatting and control character translation (as well as the input and output operations) are done by routines executing a Data Set Processing subtask of the writing task.

Initialization

When a routine of the System Output Writer reads a DSB from the output queue entry, control is passed to the DSB Handler routine (module IEFSD085). When it is entered, the DSB Handler obtains main storage for the input data set DCB (which is filled in when the data set is opened), and builds a new TIOT for the writer. The new TIOT is a two-entry TIOT; the entry for the output data set is copied from the old TIOT, and the entry for the input data set is created from information in the DSB, and in the JFCB for the input data set.

If SMF is included in the system, the DSB contains job log information. The DSB Handler determines whether this information is present, and if so begins construction of an SMF record by storing the job log information in an SMF record area.

If a system restart has been performed since the DSB was created, the volumes containing system output data sets may have been remounted on different units. The DSB handler routine uses the Queue Management Read/Write routine to bring the JFCB into main storage; the volume information is extracted, and the volumes searched for the data set. If the data set cannot be found, the routine uses the WTO macro instruction to inform the operator.

If the DSB was preceded by an SMB, or if it is the first record in the queue entry the routine obtains main storage from subpool 255 for a new CSCB, for an ATTACH macro instruction parameter list (which contains an ECB), and for an ECB list that points to the attach ECB and the cancel ECB in the subtask CSCB. The DSB handler moves the writer name into the CSCB jobname field, and issues SVC 34 to cause the Command Scheduling routine to place the CSCB in the chain of CSCBs. Thus, the program that processes the data set, whether it is the standard writer or a user-supplied program, can be canceled by operator command.

Next, the DSB handler determines whether a User routine was specified to process this data set (if so, it stores the program name), or whether the processing is to be done by the standard writer. The routine then determines from the writer CSCB whether the writer is to pause before processing any data set, or whether it is to pause only when the form specification changes. If a pause is required, the DSB handler constructs an SMF Forms record (type 6) and issues SVC 83 to have the record transferred to the SMF buffer. It then uses the WTOR macro instruction to inform the operator, and waits for the reply. When the reply is received, the DSB handler passes control to the Attachor routine.

The Attachor routine (module IEFSD070) uses the ATTACH macro instruction to pass control to either the Standard Writer routine or to the user's program. This subtask has a priority one less than that of the writing task, which enters the wait state until the attach ECB or the cancel ECB (in the subtask CSCB) is posted.

If either ECB is posted, processing of the data set has been terminated. The Attachor routine determines whether the command chaining access method is being used. If so, the Attachor passes control to the Command Chaining Access Method routine, which truncates the CCW chain and writes any filled but unwritten buffers to the output data set. On the return, the Attachor routine passes control to the Data Set Delete routine.

The Data Set Delete routine (module IEFSD171) makes the SMB Processor ready for reinitialization, determines which routine should get control, and passes control to that routine.

When it is entered, the Data Set Delete routine releases the main storage occupied by the two-entry TIOT constructed in the DSB Handler routine, then restores the TIOT pointers so that they point to the original one-entry TIOT. The routine then examines the return code passed to it by the Attachor routine.

On a normal return from the Attachor, the Data Set Delete routines uses the SCRATCH macro instruction to delete the data set (except when the data set is the

System Log data set). If it is unable to delete the data set, the routine issues a message to the operator, and passes control (and an error return code) to the Linkor routine (module IEFSD078).

If the data set was successfully deleted, the Data Set Delete routine uses the Queue Management Read/Write routine to bring the next record of the entry into main storage, examines the record, and does one of the following:

- If the record is a DSB, the Data Set Delete routine passes control to the DSB Handler routine (module IEFSD085).
- If the record is an SMB, the Data Set Delete routine issues SVC 34 to delete the subtask CSCB (command scheduling control block) from the CSCB chain, and to free the main storage it occupies. The routine then frees the main storage occupied by the ATTACH macro instruction parameter list and passes control to the SMB Handler routine (module IEFSD086).

If there are no more records in the queue entry, the Data Set Delete routine issues SVC 34 to delete the subtask CSCB from the CSCB chain, and to free the main storage it occupies. The routine then frees the main storage occupied by the ATTACH macro instruction parameter list and passes control to the Linkor routine (module IEFSD078).

If the return code from the Attachor routine indicates that a permanent I/O error occurred in the input data set, the Data Set Delete routine issues a message to the operator, but otherwise performs the same processing it performs on a normal return.

If the return code from the Attachor routine indicates that a permanent I/O error occurred in the output data set, the Data Set Delete routine notifies the programmer that the processing of the entry has been interrupted. If at least one queue entry record has been successfully processed, the routine builds an SMB, places a message in it, and makes the SMB point to the DSB corresponding to the data set whose processing was interrupted. The Data Set Delete routine then uses the Queue Management Read/Write routine to write the SMB as the first record of the current queue entry. Thus, when the queue entry is retrieved for processing, the SMB constructed by the Data Set Processor will be processed first. Since the SMB points to the DSB corresponding to the data set whose processing was interrupted, intervening records will not be processed; the data set will be reprocessed, and subsequent records will be retrieved in order.

When the SMB has been written, the Data Set Delete routine issues SVC 34 to delete the subtask CSCB from the CSCB chain, and to free the main storage it occupies. The routine then frees the main storage occupied by the ATTACH macro instruction parameter list and passes control to the Linkor routine (module IEFSD078).

The Data Set Processing Subtask

The Data Set Processing Subtask includes the functions of obtaining records from the input data set, of moving records from input buffers to output buffers, of reformatting and translating control characters as necessary, and of placing records in the output data set. The term "records", as used in this section, refers to the logical records handled by the GET/PUT (LOCATE) macro instructions. In the case of a data set consisting of VS (variable-length spanned) records, a more precise term would be "segments."

The first routine entered to perform the Data Set Processing subtask is the Standard Writer routine (module IEFSD087). When it is entered via the ATTACH macro instruction, if the output device is a UCS printer, prior to opening the input data set the Standard Writer routine issues the LINK macro instruction for the 3211 Processor Module (IEFSDTTE). This processor routine analyzes the input JFCB for user UCS/FCB environmental change requests and dynamically loads the UCS/FCB buffers for each input data set. On return from the 3211 Processor routine, the Standard Writer routine uses the OPEN (type J) macro instruction to open the input data set. If the input data set was never opened by the job step (thus contains no records), a DCB exit routine sets a switch. On the return from the Open routine, the Standard Writer tests the switch; if the data set was not properly opened, the Standard Writer routine returns control to the Attachor routine, thus terminating the task.

If the data set was properly opened, the Standard Writer routine examines the input DCB to determine whether the input data set contains VS records. If so, and if the Spanned Data Sets routine (module IEFSDXXX) has not already been loaded into main storage, the Standard Writer routine obtains main storage for control information and issues the LOAD macro instruction. When the routine has been loaded (or if it was already in main storage) the Standard Writer routine passes control to the Spanned Data Sets routine for initialization.

On the return from the Spanned Data Sets routine, the Standard Writer routine tests for any initialization errors. If an error occurred, the routine uses the WTO macro instruction to inform the operator, then closes the input data set, executes the Transition routine, and returns control to the Attachor routine. If no error occurred, or if the Spanned Data Sets routine is not to be used, the Standard Writer routine passes control to the Transition routine.

On the return from the Transition routine (module IEFSD088), the Standard Writer routine begins its main-line processing. The routine obtains a record (or a segment, if VS records are being processed) from the input data set via a GET macro instruction (locate mode). The routine stores the control character, segment information, and record length, then resets the pointer to the record so that it points to the first data character instead of to the control character. Finally, the Standard Writer passes control to the Put routine, and on the return, issues another GET macro instruction.

The Standard Writer routine continues the process of obtaining a record, storing control information, and passing control to the Put routine, until either an end-of-data or a permanent I/O error condition is encountered in the input data set. In either case, the Standard Writer routine passes control to the Transition routine; then closes the input data set, frees the main storage obtained in the Spanned Data Sets routine, and returns control to the Attachor routine.

PROCESSING SYSTEM MESSAGE BLOCKS

The SMB Processor extracts the messages from SMBs and writes the messages to the writer's output data set. The SMB Processor gets control whenever the Linkor routine (module IEFSD078) determines that the record read in the Main Logic Control routine is an SMB, and whenever the Data Set Delete routine (module IEFSD171) of the DSB Processor reads an SMB. It returns control to the Linkor when there are no more queue entry records to process, and it passes control to the DSB Handler routine (module IEFSD085) of the DSB Processor when the next record to be processed is a DSB.

The first module of the SMB Processor to be executed is the SMB Handler routine (module IEFSD086). When it is first entered, the SMB Handler examines the output UCB device type. If it is a 3211 UCS printer, the SMB handler determines if the currently loaded UCS/FCB image-ids are default images and if data checks are blocked. If not the SMB handler issues the LINK macro instruction to the 3211 Processor Module (IEFSDTTE) to load the writer default image-ids saved in the PARLIST parameter list by the Initialization routine (IEFSD080) and blocks data checks during SMB processing. On return from the 3211 Processor routine, the SMB Handler inspects the output data set DCB to determine whether the output data set is to contain variable-length spanned (VS) records. If so, the SMB Handler routine passes control to the Spanned Data Sets routine (module IEFSDXXX) for initialization.

On the return from the Spanned Data Sets routine, or if it determines that VS records are not to be processed, the SMB Handler passes control to the Transition routine (module IEFSD088) to ensure that overprinting will not take place. On the return, the SMB Handler extracts the messages from the SMB.

If a message is compressed, the SMB Handler expands it before storing a pointer to its location. When the pointer has been stored, the SMB Handler passes control to the Put routine (module IEFSD089). This routine uses the PUT macro instruction to place the message in the output data set.

On the return from the Put routine, the SMB Handler extracts the next message from the SMB and passes control back to the Put routine. When there are no more messages to be extracted, the SMB Handler passes control to the Transition routine.

On the return from the Transition routine, the SMB Handler uses the Queue Management Read/Write routine to bring the next record from the queue entry into main storage. If the record is an SMB, the SMB Handler executes the Transition routine and continues passing the messages to the Put routine. If the record is a DSB, or if there are no more records in the queue entry, the SMB Handler determines whether the writer is using the command chaining access method. If so it passes control to the Command Chaining Access Method routine to write out any filled (but unwritten) buffers. On the return, it returns control to the Linkor routine (module IEFSD078).

If the return from the Put routine indicates that a permanent I/O error has been encountered, the SMB Handler routine returns control to the Linkor routine (module IEFSD078).

SERVICE ROUTINES

Because of the similarities in the processing performed by the Separator program, by the Data Set Processor and by the SMB Processor, they use certain routines in common:

- The Put routine (module IEFSD089), which is entered from the Standard Writer, Transition, and SMB Handler routines. The Put routine performs control character translation, moves records and messages from input buffers to output buffers (or uses the Spanned Data Sets routine to move segments), and issues the PUT macro instruction. If SMF is included, the Put routine increments the temporary output record counter in the SMF record for each record written to the output device.

- The Spanned Data Sets routine (module IEFSDXXX) which is entered from the Put routine. The Spanned Data Sets routine moves variable-length spanned (VS) record segments from input buffers to output buffers.

- The Command Chaining Access Method routine (module IEFSDXYZ), which is entered via the PUT macro instruction. The Command Chaining Access Method routine simulates the processing performed by the QSAM Put Locate and Truncate routines, except that it uses command chaining.

- The Transition routine (module IEFSD088) which is entered from the Standard Writer routine and the SMB Handler. The Transition routine prevents overprinting from occurring in the transition between queue entry records.

- The 3211 Processor routine (module IEFSDTTE) which is entered via the LINK macro from the Standard Writer routine, the SMB Handler, or the Job Separator routine. The 3211 Processor routine initializes the UCS/FCB buffers for 3211 UCS printer output prior to the output of each data set, group of SMBs or job separator.

The Put Routine

The Put routine (module IEFSD089) is entered from the Standard Writer routine, the SMB Handler routine, the Job Separator routine, or the Transition routine when a record (or segment) is available in an input buffer or work area. When it is entered, the routine determines whether control character processing is necessary, and if so, performs the necessary processing.

The control characters associated with a logical record (or with the first segment of a VS record) are carriage tape channel numbers, line spacing indicators, or punch stacker numbers. An input data set may or may not use control characters. If it does, they are expressed as American National Standard control characters, or as machine control characters. Similarly, the output data set may use either American National Standard or machine control characters, or none. Since American National Standard control characters are acted upon before a record is printed or punched, and machine control characters are acted upon afterward, the Put routine may have to translate a control character and associate it with a record other than the one it originally accompanied. The kind of processing required depends on the combination of control character types used in the input and output data sets, and is shown in Figure 22.

When it has performed any necessary control character processing, the Put routine determines whether it must process VS records.

If VS records are to be processed, the Put routine uses the Spanned Data Sets routine (module IEFSDXXX) to move the segments from the input buffers to the output buffers. The Put routine passes control to the Spanned Data Sets routine with an indication (in the high-order byte or Register 15) that a segment is available. The Spanned Data Sets routine passes a return code, which is examined by the Put routine when it regains control, that indicates whether another segment is required or whether a PUT macro instruction should be issued.

If the records to be processed are not VS records, the Put routine does the moving from input buffer to output buffer. If the input record is too large for the output buffer, the Put routine truncates the record at the low-order end. If a fixed-length record is smaller than the buffer, the Put routine left justifies the record and pads with blanks.

If a permanent I/O error occurs, the Put routine returns control to its caller.

Input Data Set Control Characters	American National Standard	X	X	X						
	Machine				X	X	X			
	None							X	X	X
Output Data Set Control Characters	American National Standard	X			X			X		
	Machine		X			X			X	
	None			X			X			X
DO ACTION#		1	2	4	3	1	4	5	6	1

Actions:

1. No control character processing.
2. Use the control character from the current record for the previous record.
3. Use the control character from the previous record for the current record.
4. Do not use control characters for output records.
5. Generate an American National Standard control character for each output record.
6. Do Action 5, then do Action 2.

Figure 22. System Output Writer Control Character Processing

The Spanned Data Sets Routine

The Spanned Data Sets routine (module IEFSDXXX) is used by the Put routine when either the input data set or the output data set (or both) consists of variable-length spanned (VS) records. The routine is entered each time an input buffer (or work area) becomes available, and each time a PUT macro instruction makes an output buffer available. It moves records (or segments) from the input buffers (or work area) to the output buffers. If the input data set has American National Standard control characters and the output data set has machine control characters, the routine uses a work area tơ hold a complete record until the control character from the next record is available.

If the work area is not required, the Spanned Data Sets routine moves the contents of the input buffer directly to the output buffer. When the output buffer is full, the routine returns control to the Put routine to have a PUT macro instruction issued. When the input buffer is empty, the Spanned Data Sets routine returns control to the Put routine, which returns control to its caller to obtain another buffer.

If the work area is used, the Spanned Data Sets routine fills it from the input buffer. The work area is as large as the largest record to be handled by the routine, and if more than one input buffer is required to fill the work area, the routine returns control to the Put routine with a GET request. The Put routine returns control to its caller, which obtains another input buffer (or generates another record), and passes control to the Put routine. The Put routine then passes control to the Spanned Data Sets routine, which moves the contents of the input buffer (or the generated record) to the work area.

When the first segment of the next record is available, the Spanned Data Sets routine fills the output buffer from the work area, then returns control to the Put routine to have a PUT macro instruction issued. If more output buffers are required to empty the work area, the routine continues to request PUTs until the work area is empty.

The Command Chaining Access Method Routine

The Command Chaining Access Method routine (module IEFSDXYZ) simulates the processing performed by the QSAM Put Locate and Truncate routines. The difference is that the Command Chaining Access Method routine uses command chaining to write one string of up to nine buffers with a single IOB and a single EXCP macro instruction. If ten or more buffers are specified, the routine writes them as two strings, but uses one IOB.

The Command Chaining Access Method routine is entered under the following circumstances:

- Via a PUT macro instruction issued in the Put routine (module IEFSD089).
- Via a branch instruction from the Attachor routine (module IEFSD070) after each data set has been processed.
- Via a branch instruction from the SMB Handler routine when it determines that the next record to be processed is a DSB, or that there are no more records in the queue entry.
- From the Linkor routine (module IEFSD078) after the Separator program has been executed.

When it is entered via a PUT macro instruction, the Command Chaining Access Method routine determines whether all of the buffers in a chain have been filled, and if so it writes them to the output data set. In any case, it returns control to its caller with a pointer to the next available buffer.

When it is entered via a branch instruction, the routine breaks the CCW chain at the CCW corresponding to the last buffer filled. It then uses the EXCP macro instruction to write the filled buffers to the output data set, and returns control to its caller.

The Transition Routine

The Transition routine (module IEFSD088) is used by the DSB Processor, the SMB Processor, and the Separator program to perform the following functions:

- It prevents overprinting from occurring when a data set using American National Standard control characters is followed by a data set using machine control characters.

- It causes extra records to be printed or punched at the beginning of a data set so that the initial control character will be effective.

- It causes extra records to be printed or punched at the end of a data set so that the final control character will be effective.

- It causes extra records to be printed or punched at the end of a data set or a group of messages so that the last significant record will be written before a PAUSE option takes effect.

The Transition routine is entered from the Standard Writer routine before a data set is processed and again when an end-of-data condition is encountered. It is entered from the SMB handler routine before and after each SMB is processed, and it is entered from the Separator program before and after the separator records have been written.

When it is entered, the Transition routine determines the type of processing required by making the decisions shown in Figure 23, then performs the required processing. Each time it is required to write a record to the output data set, the Transition routine uses the Put routine (module IEFSD089). When it has completed its processing, the routine returns control to its caller.

3211 Processor Routine (IEFSDTTE)

This module is entered only if the output device is a 3211 Printer and some change to the UCS/FCB environment has been predetermined. A complete analysis is performed to identify those parameters requiring change as determined from caller requests for parameters which differ from the presently established environment. The input JFCB entry is first checked for a change request. This is followed by a test for valid association with prerequisite parameters or the need to secure default parameters. The input request is then compared with the present parameters as defined in the output UCB. If the input request differs from the present output, the input request is transferred either from the input JFCB or from PARLIST (in the case of default parameters) to the SETPRT list in core. The SETPRT macro is then executed to permit data management to reload the UCS/FCB buffer(s), block data checks, load character set images in the folded mode or request operator verification of images or alignment of forms. The return codes in register 15 reflecting the status of the UCS/FCB load are used as offsets into a branch control table to select an input error message for sysout to the printer or an output message for a WTO to the operators' console. The System Output Writer PUT routine in IEFSD089 is used to output the error message to the printer. If an input error occurred, a switch is set in the WKSOR communication region to forward a request for input data set skip and proceed to the next input data set. An output error return code in register 15 results in writer termination.

This routine cannot be used with a user-written writer routine for the following reasons.

- It utilizes the standard 2-register interface with IEFSD089 to pass the start and length of a line of print and printer control characters (the sysout printer error message) to the Standard Writer PUT routine for output.

- It requires the use of the WKSOR work area to locate the PARLIST communication area in which default image-ids are saved and to pass input error indicators to IEFSD087 to effect skipping of data sets in error and advancement to the next data set. This WKSOR area is gotton and initialized by the Standard Writer Routine IEFSD087.

A user-written writer routine has the SETPRT macro facility available whereby the required UCS/FCB environment can be user established at will.

IEFSDTTE is entered from:

- IEFSD086 for loading of default images and to block data checks for SMB's.
- IEFSD087 to process UCS/FCB environmental changes for each input data set.
- IEFSD094 for loading default images. Control is returned to the calling routine or passed to IEFSD089 to output error messages.

Output for Printing		Y	Y	Y	Y	Y	Y	Y	Y		
Output for Punching										Y	Y
Input Control Character	American National Standard	Y	Y	Y	Y						
	Machine					Y	Y	Y	Y		
Output Control Character	American National Standard	Y				Y	Y			Y	
	Machine		Y	Y				Y			Y
	None				Y				Y		
Start Transition			Y			Y					
End Transition		Y		Y	Y		Y	Y	Y	Y	Y
Do Action #		1	2	3	4	5	6	6	4	7	8

Actions:

1. Print a line of blanks with American National Standard Suppress c.c.
2. Print a line of blanks with American National Standard Space 1 c.c.
3. Put a Machine Space 1 c.c. in the previous record and do Action 6.
4. Print a line of blanks with no c.c.
5. Put an American National Standard Space 1 c.c. in the current record.
6. Print a line of blanks with Machine Suppress c.c.
7. Punch 3 blank records, each containing an American National Standard Stacker 1 c.c.
8. Punch 3 blank records, each containing a Machine Stacker 1 c.c.

Figure 23. Transition Routine Decision Table

Part 5: Command Processing

Command processing consists of command scheduling and command execution. Command scheduling is the synchronization of command execution with other events in the system; it is accomplished when the Command Scheduling routine (SVC 34) stores the command, and notifies a job management routine that the command is available for execution. Command execution is accomplished by various Job Management routines as part of their performance of existing system tasks, or by Job Management routines performing tasks established in response to the command.

Operator commands may be read from any of the units designated as console input devices, they may be read from a TSO terminal, or they may be placed in an input stream and read by an interpreter as a part of a reading task. In addition, commands may be assembled into the system when it is generated; these commands are executed after initial program loading, as part of system initialization.

The reading of commands from console devices is described in the MVT Supervisor Program Logic Manual. The reading of commands from a TSO terminal is described in the TSO Command Processor Program Logic Manuals. The reading of commands by the interpreter is described in the section on The Interpreter Routine in this publication.

The "Command Scheduling" portion of this section discusses the initialization of the Command Processing routine (SVC 34) and the techniques it uses for storing commands, notifying the responding routine that a command is to be processed, and issuing error messages. The "Command Execution" portion discusses the commands themselves: the way they are processed in the SVC 34 routine, and the way they are processed by special command execution tasks.

Command Scheduling

Commands must usually be executed at distinct points in the cycle of events in the system. Since the operator cannot synchronize the execution of his commands with the system, the execution of commands must be scheduled by the system. Command scheduling is the synchronization of command execution with task processing. Commands must first be stored--compactly, conveniently, and chronologically. An appropriate system routine must then be notified that a command is awaiting execution.

The Command Scheduling routine (SVC 34) condenses and stores command verbs and operands, and notifies the appropriate responding system routine. Many commands are stored in command scheduling control blocks (CSCBs) that are chained together; the Command Scheduling routine manipulates the CSCB chain as required.

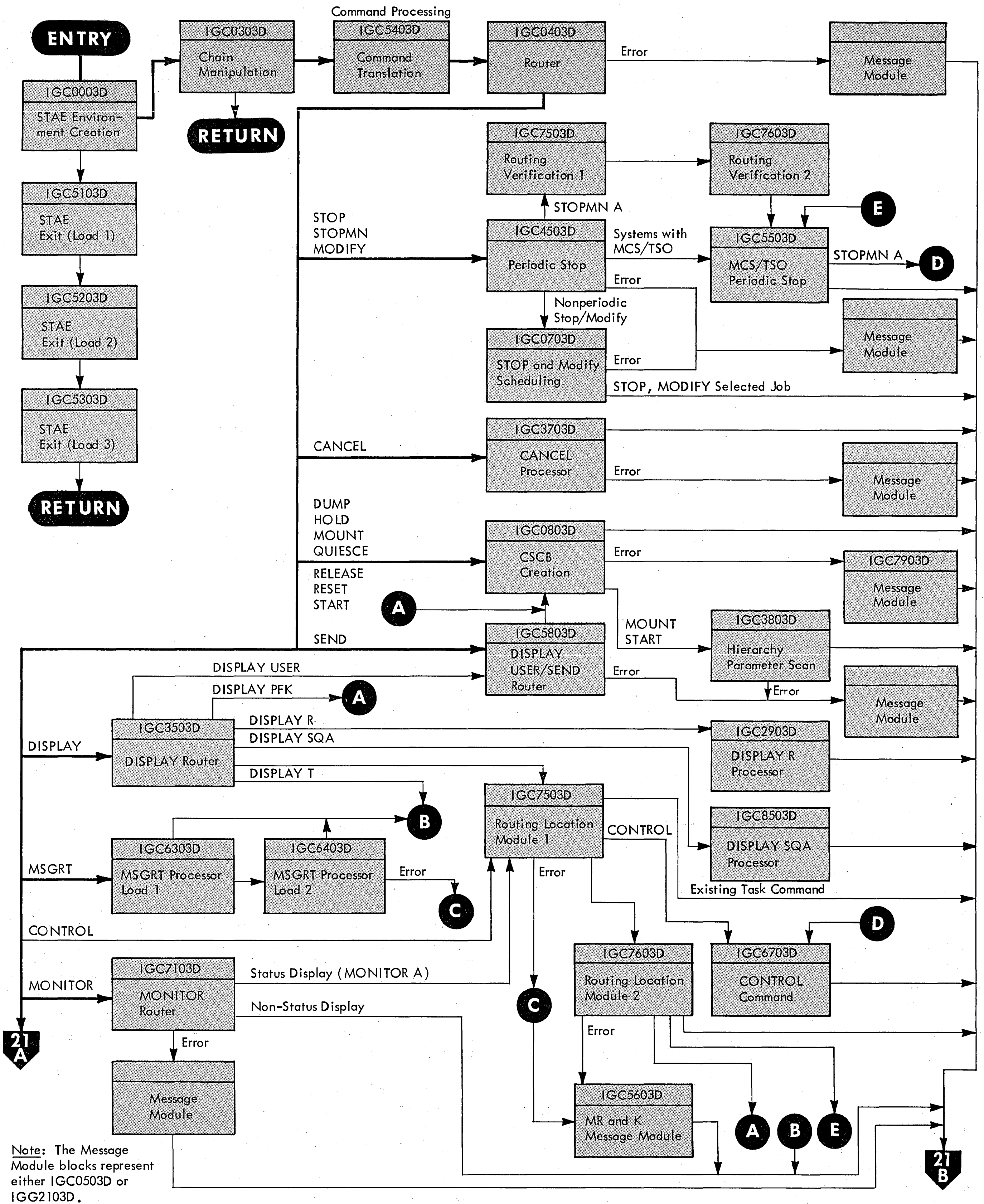
The Command Scheduling routine, shown in Figures 24 and 45, is entered whenever an SVC 34 is issued by routines performing the console and TSO terminal communications, reading, initiating, and writing tasks. The routine is used whenever there is an operator command ready for scheduling, whenever an executing job step is to be canceled, and whenever a CSCB must be deleted from the chain and the CIB or, in the case of TSO, a LOGON communications element (LCE), is to be added or deleted from the chain. This section discusses only the command scheduling functions; the other functions are discussed in the section on the initiating task.

INITIALIZING THE COMMAND SCHEDULING ROUTINE

The first module of the Command Scheduling routine (SVC 34) to be executed is the STAE Environment Creation routine (module IGC0003D).[1] (For a detailed discussion of the STAE facility, see the MVT Supervisor PLM.) This routine is entered twice each time SVC 34 is issued. When it is entered following the issuing of SVC 34, the routine creates the STAE environment. When a failure occurs during execution of the command or master scheduling tasks, the routine is entered again. This time it passes control to the STAE Exit routine to prepare for retry.

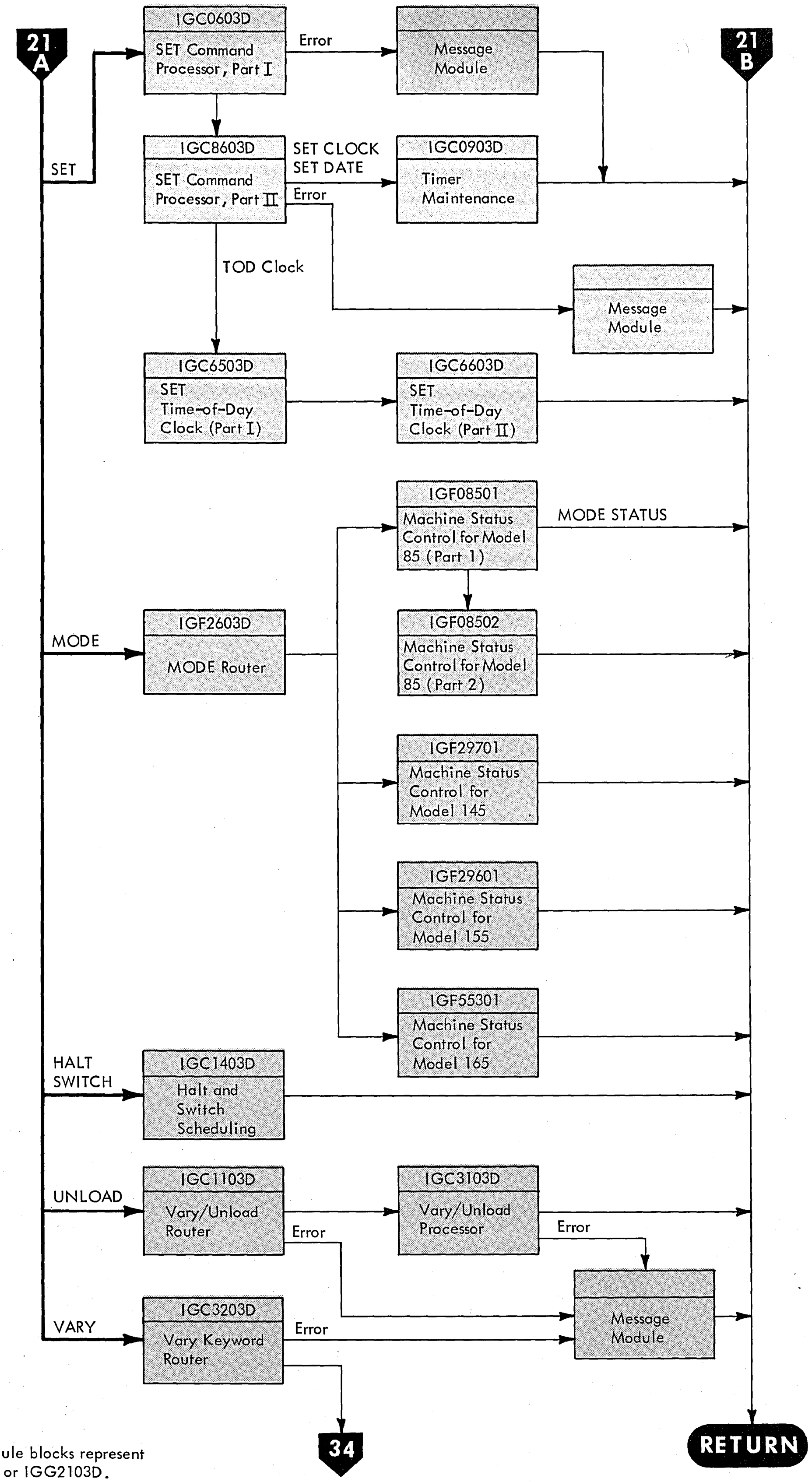
When the routine receives control, it first checks register 1. If register 1 points to the characters "STAE", the routine has been entered because of a

[1] Throughout the discussion of the Command Scheduling routine (SVC 34), load module names will be used in order to indicate transient SVC routines that are used by the master scheduler.



Note: The Message Module blocks represent either IGC0503D or IGG2103D.

Figure 24. Control Flow in the Command Scheduling Routine (Part 1 of 2) (Part I -- See Figure 45 for Part II)



Note: The Message Module blocks represent either IGC0503D or IGG2103D.

Figure 24. Control Flow in the Command Scheduling Routine (Part 2 of 2) (Part I -- See Figure 45 for Part II)

failure during execution of the command or master scheduling tasks. Therefore, the routine uses an XCTL macro instruction to pass control to the first load of the STAE Exit routine (module IGC5103D).

If register 1 does not point to "STAE", the routine checks register 0. If register 0 does not point to "STAE", this is the first entry to the routine for this issuing of SVC 34. The routine saves registers 0, 1, 5, and 15 in the extended save area (XSA) of the supervisor request block (SVRB) that was created when SVC 34 was issued. Then the routine places "STAE" in the XSA and places the address of "STAE" in register 0. Finally, the routine issues SVC 34.

If register 0 points to "STAE", this is the second entry to the routine for this issuing of SVC 34. The routine establishes the STAE environment by placing the STAE Exit Linkage routine in the XSA. If SVC 34 was issued in response to a command, the routine stores the command verb with the STAE Exit Linkage routine in the XSA. Then, the routine restores registers 0, 1, and 5 to their original condition upon entry to the routine and places the contents of register 15 in the sixth word of the second XSA. Finally, the routine passes control via an XCTL macro instruction to the Chain Manipulation routine (module IGC0303D).

When the Chain Manipulation routine is entered, it tests register 1:

- If register 1 is positive, SVC 34 has been entered to schedule the execution of a command. In this case, the Chain Manipulator routine uses the XCTL macro instruction to pass control to the Command Translator routine (module IGC5403D).

- If register 1 is not positive, the routine tests both register 1 and register 0 and performs the indicated processing as follows:

Register 1	Register 0	Processing
Negative	Zero	CSCB processing
Negative	Negative	Set CIB count in CSCB
Negative	Positive	CIB processing
Zero	Positive	Free CIB chain
Zero	Negative	Set CIB count to zero

When the Command Translator routine is entered to schedule the execution of a command, register 1 points to a parameter list (on a halfword boundary) consisting of a length field and a buffer containing the command, and register 0 indicates the source of the command.

- If register 0 is positive and the high order bit of the lower half word is on, register 0 contains the ID of the TSO terminal through which the command was issued (TJID).

```
Register 0  ┌──────────┬───┬──────────┐
            │          │ 1 │   TJID   │
            └──────────┴───┴──────────┘
             31         15           0
```

The Command Translator routine places the terminal ID in the TJID field of the extended save area (XSA) of the supervisor request block (SVRB) -- see Figure 26.

- If register 0 is positive and the high order bit of the lower half word is not on, register 0 contains the UCMI corresponding to the console through which the command was issued, or a X'00' showing that the command was issued by a system routine.

```
Register 0  ┌───────┬───┬─────────────┐
            │       │ 0 │UCMI or X'00'│
            └───────┴───┴─────────────┘
             31      15              0
```

In systems with the multiple console support (MCS) option, the Command Translator routine determines whether the command is to be added to the hard copy log. If so, it issues a WTO macro instruction with the MCSFLAG=HRDCPY parameter, and stores the contents of register 0 in the UCMI field of the XSA.

In systems without MCS, the routine stores zeros in the UCMI field of the XSA.

- If register 0 is negative, the SVC was issued in the Interpreter routine. In this case, the register contains the command authorization field from the PARM field of the reader procedure EXEC statement.

In systems with MCS, the Command Translator routine determines whether the command is to be added to the hard copy log. If so, it issues a WTO macro instruction with the MCSFLAG=HRDCPY parameter and stores a X'80' in the UCMI field of the XSA. Then it stores the contents of the register in the module interface work area.

In systems without MCS, the routine stores a X'80' in the UCMI field of the XSA.

When it has updated the XSA, the routine translates any lowercase EBCDIC Characters in the command (except those enclosed in parentheses) into uppercase EBCDIC characters and uses the XCTL macro instruction to pass control to the Router routine.

The Router routine (module IGC0403D) scans the command buffer for the command verb, and uses the verb to find the appropriate entry in the verb table (see Figure 25). If there is no verb table entry corresponding to the command verb, the command is invalid; the Router routine uses the XCTL macro instruction to pass control to a message routine that informs the operator and returns control to the supervisor.

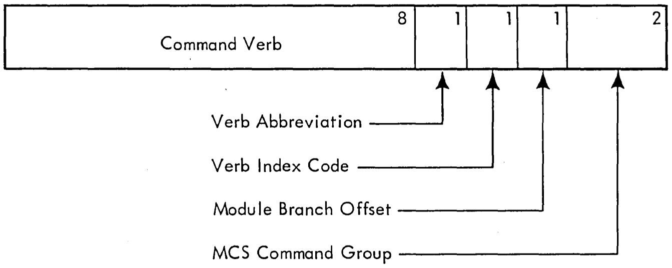


Figure 25. Verb Table Entry

If the command is valid, the Router routine determines whether the source has the authority to issue the command. If the UCMI field of the XSA contains X'00' (the source is a system routine or the system does not contain the MCS option) any valid command is authorized. If the UCMI field contains X'80', the command source is the input stream, and the command authorization is in the module interface work area of the XSA. The routine compares the authorization to the MCS command group field in the verb table entry. Otherwise, the routine uses the UCMI to find the appropriate unit control module entry, and compares the command authorization to the MCS command group field in the verb table entry.

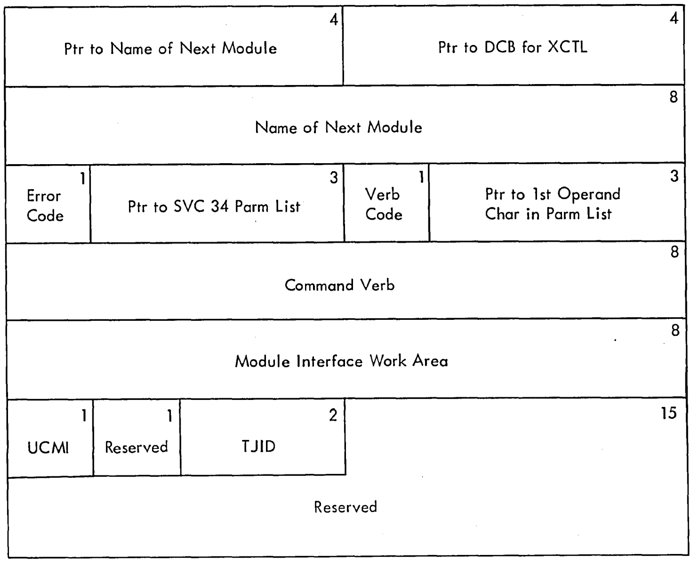


Figure 26. Extended Save Area of SVRB as Used for Passing Information Between Modules of the SVC 34 Routine

If the source is not authorized to issue the command, the Router routine uses the XCTL macro instruction to pass control to a message module that informs the issuing operator and returns control to the supervisor.

If the source is authorized to issue the command, the Router routine stores a X'00' in the UCMI field of the XSA (unless the command is a VARY command) and uses the XCTL macro instruction to pass control to the module specified in the verb table entry.

If the command is a VARY command, the authorization necessary depends on the second operand; the router routine therefore assumes that no authorization is necessary, and the authorization is checked in a subsequent module.

STORAGE AND NOTIFICATION

The Command Scheduling routine uses several techniques for storing commands and notifying a responding routine. Figure 27 lists the commands alphabetically (within storage technique), and also shows the method of notification and the name of the responding routine. The storage techniques are discussed below:

- Creating a New CSCB: When the execution of a command as part of an existing system task is not feasible (see task-creating commands), a new command execution task must be established. The CSCB Creation routine (module IGC0803D) determines whether the command is one that will be executed by the master scheduler (DISPLAY A, DISPLAY U, DISPLAY CONSOLES, DISPLAY C,K, DISPLAY M, DISPLAY PFK, MONITOR A, QUIESCE, VARY CH, VARY CPU, and VARY STOR). If the command is not one of these commands, the routine builds a CSCB to contain the command, encodes the command, and stores it in the CSCB.

  If the command is one of those executed in the master sheduler region, the CSCB Creation routine determines whether it can be executed. Because of storage limitations, only one such command can be executed at a time, except that in a system that includes multiprocessing as many as two VARY STOR OFFLINE commands can be executed concurrently with one other command.

  If the command cannot be executed, the CSCB Creation routine passes control to a message routine that informs the operator and returns control to the supervisor. If the command can be executed, the CSCB Creation routine builds a CSCB to contain the command, encodes the command, and stores it in the CSCB.

  When a CSCB has been constructed, it is marked "pending" and added to the chain of CSCBs, where it becomes a communications region between the Command Scheduling routine and the master scheduler. It is the master scheduler's task to attach system tasks; when it is notified of the command (via a POST macro instruction), it scans the chain and attaches the task of executing the command.

- Updating an Existing CSCB: When a CANCEL command is to be executed as a part of a specific, currently existing system task (a job that has been selected, a TSO job, or the subtask of a system output writer), the CANCEL Processor routine (module IGC3703D) marks the CSCB that corresponds to the task and issues a POST macro instruction to the ECB in the CSCB. The executing routine tests the bits in the CSCB, and executes the command.

- Chaining the CIB: When a command (other than the CANCEL command) is to be executed as a part of a specific, currently existing system task, the Stop and Modify Scheduling routine (module IGC0703D) builds a command input buffer (CIB) in subpool 245 from the command image and adds the CIB to the CIB chain. Routines that execute the task inspect the CIB chain and then execute the command(s).

- Using a System Table: When the command is to be executed by a specific routine (performing any existing system task), the operands are encoded and stored in system tables. The responding routine tests bits in the tables during its normal operating cycle, and executes commands as indicated.

- Using General Registers: When the Execution routine can operate as a part of the Command Scheduling routine, and when the information in the operands of the command can be placed in general registers, this is the technique used. Notification consists of passing control to the Execution routine.

- Building a Parameter List: When the Execution routine can operate as a part of the same task as the Command Scheduling routine, but the operand information cannot be placed in general registers, a parameter list is constructed. Notification is accomplished by passing control to the responding routine.

Storage	Command	Notification	Initial Responder
Create new CSCB	CANCEL (job in queue) DISPLAY A DISPLAY C,K DISPLAY CONSOLES DISPLAY jobname DISPLAY M DISPLAY N DISPLAY PFK DISPLAY Q DISPLAY U DISPLAY USER[3] DUMP HOLD MONITOR A MOUNT QUIESCE RELEASE RESET SEND[3] START VARY CH VARY CPU VARY STOR	POST	Master Scheduler
Update CSCB	CANCEL (job selected/ TSO session)		Initiator
	CANCEL wtrdevice		Writer
Chain CIB	MODIFY STOP		Named task
Update system tables	MONITOR DSNAME MONITOR JOBNAMES MONITOR SESS[3] MONITOR SPACE MONITOR STATUS	Inspect tables	Initiator

	MODE
	SET PROC
	SET Q		

	STOPMN[4] DSNAME STOPMN[4] JOBNAMES STOPMN[4] SESS[3] STOPMN[4] SPACE STOPMN[4] STATUS UNLOAD VARY unit		Initiator
	SWITCH	SVC 83	SMF SVC Routine
	SWAP[2] VARY PATH[2]	POST	Master Scheduler

[1] See the MVT Supervisor Program Logic Manual.

[2] See the Input/Output Supervisor Program Logic Manual.

[3] See the TSO Command Processor Program Logic Manuals.

[4] See the "Summary of Amendments" for Release 21.

Figure 27. Command Storage and Notification Techniques (Part 1 of 2)

Storage	Command	Notification	Initial Responder
Load general registers	DISPLAY R		Timer Maintenance routine[1]
	DISPLAY SQA		
	DISPLAY T	TIME	
	SET CLOCK	STIMER	
	SET DATE	XCTL	
Build parm list	HALT[1]	SVC 76	Statistics Update routine[2]
	LOG	WTL	Log routine[1]

[1] See the MVT Supervisor Program Logic Manual.
[2] See the Input/Output Supervisor Program Logic Manual.
[3] See the TSO Command Processor Program Logic Manuals.
[4] See the "Summary of Amendments" for Release 21.

Figure 27. Command Storage and Notification Techniques (Part 2 of 2)

ERROR MESSAGE PROCESSING

Validity checking is performed in all modules of the Command Scheduling routine. If an error condition is encountered in any module, a code representing the error is placed in the extended save area of the SVRB (shown in Figure 28) and control is passed to the appropriate Error Message routine (module IGC0503D or module IGG2103D). The routine uses a WTO macro instruction, or a TPUT macro instruction if the command was issued from a TSO terminal, to inform the operator of the error condition, then returns control to the supervisor.

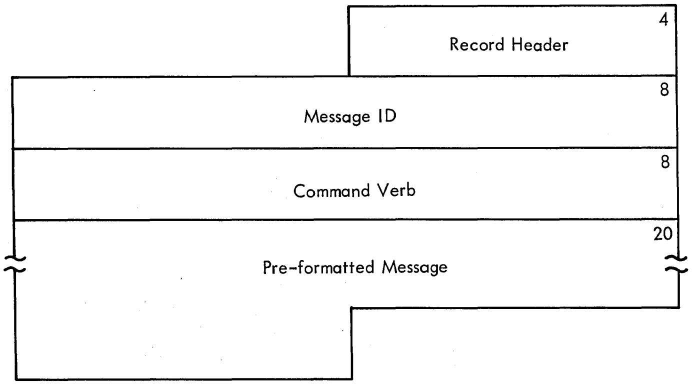


Figure 28. Extended Save Area of SVRB as Used for Message Processing in the SVC 34 Routine

STAE EXIT ROUTINES

If a STAE environment is in effect and a failure occurs during the command or master scheduling tasks, ABTERM processing obtains the address of the STAE Exit Linkage routine from the STAE control block and passes control to the routine. The Exit Linkage routine issues SVC 34 with register 1 pointing to the characters "STAE" in main storage.

When the first routine (module IGC0003D) of SVC 34 receives control, it checks register 1. When it finds that the register points to "STAE", it uses an XCTL macro instruction to pass control to the first load of the STAE Exit routine (modules IGC5103D, IGC5203D, IGC5303D).

If the SYS1.DUMP data set is available, module IGC5103D provides a dump of the entire main storage by means of the SVC dump facility. Then it sets a code for the third load module of the STAE Exit routine (module IGC5303D), which will issue a message indicating that the dump was provided.

Next, module IGC5103D scans the following control blocks for errors: the command scheduling control block (CSCB), the command input buffer (CIB), the group control block (GCB), and the LOGON communication element (LCE). If a chain pointer is found to be in error (i.e., it points out of the system queue area or to other than a doubleword boundary), the routine truncates the chain by setting the pointer to zero. The next pointer is then checked. When all the control blocks have been checked, module IGC5103D determines whether an ENQ macro instruction had been issued specifying the GCB chain. If so, it releases the chain using a DEQ macro instruction. Then module IGC5103D passes control to module IGC5203D.

Module IGC5203D scans the reply queue element (RQE) chain and truncates it if any block in the chain is not in the SQA or is not on a doubleword boundary. Next, if the Display Active routine (module IEEVDSP1) was processing at the time of the failure, module IGC5203D frees the main storage used by the routine and turns off the display active bit in the M/S resident data area. After all of this processing is completed, module IGC5203D passes control via an XCTL macro instruction to the third load of the STAE Exit routine (module IGC5303D).

For each error that modules IGC5103D and IGC5203D find, module IGC5303D issues a message to the console indicating the failing component, the error that was found, and the action, if any, that was taken. The routine also issues a message indicating the command that was being processed at the time of the failure, if the command can be determined. If the command was from a TSO terminal, the routine issues the messages to the terminal as well as to the console.

If the error occurred during the master scheduling task, module IGC5303D returns control to the Master Scheduler STAE Retry routine through the ABEND processing. If the error occurred during the command scheduling task (SVC 34), the routine places the Command Scheduler STAE Retry routine in the XSA and passes control to it via ABEND.

STAE RETRY ROUTINES

The Command Scheduler STAE Retry Routine: This routine is part of module IGC5303D. Placed in the XSA by module IGC5303D, the routine frees the main storage occupied by the ABEND/STAE work area and returns control to SVC 34.

The Master Scheduler STAE Retry Routine: This routine is part of the master scheduling task. After receiving control from ABEND processing, the routine frees the ABEND/STAE work area and branches to the entry point of the Master Scheduler Wait and Attach routine (module IEEVWAIT).

Command Execution

The execution of a command is the performance of the function specified in the command. These functions are performed by Job Management routines, either as new tasks established by the master scheduler, or as parts of existing system tasks. Commands are classified, accordingly, into two categories: task-creating commands, and existing-task commands. Figure 29 lists in alphabetical order the commands whose execution is discussed in this publication.

TASK-CREATING COMMANDS	EXISTING-TASK COMMANDS
CANCEL	CANCEL (job selected)
(job in queue)	CANCEL (TSO user)
DISPLAY A	CANCEL wtrdevice
DISPLAY C,K	CONTROL
DISPLAY CONSOLES	DISPLAY R
DISPLAY jobname	DISPLAY SQA
DISPLAY M	DISPLAY T
DISPLAY N	HALT
DISPLAY PFK	MODE
DISPLAY Q	MODIFY
DISPLAY U	MONITOR DSNAME
DISPLAY USER	MONITOR JOBNAMES
DUMP	MONITOR SESS
HOLD	MONITOR SPACE
MONITOR A	MONITOR STATUS
MOUNT	MSGRT
QUIESCE	SET
RELEASE	STOP
RESET	STOPMN[1] A
SEND	STOPMN[1] DSNAME
START	STOPMN[1] JOBNAMES
VARY CH	STOPMN[1] SESS
VARY CPU	STOPMN[1] SPACE
VARY STOR	STOPMN[1] STATUS
	SWAP
	SWITCH
	UNLOAD
	VARY ONLINE/OFFLINE
	VARY CONSOLES
	VARY HARDCPY
	VARY MSTCONS
	VARY PATH

[1] See the "Summary of Amendments" for Release 21.

Figure 29. Classification of Commands

The LOG, REPLY, CONTROL, and WRITELOG commands are discussed in the MVT Supervisor Program Logic Manual; the HALT, SWAP, and VARY PATH commands are discussed in the I/O Supervisor Program Logic Manual; the commands associated with remote job entry processing are discussed in the Remote Job Entry Program Logic Manual; the commands associated with graphic job processing are discussed in the Graphic Job Processor Support PLM. The commands associated with TSO are discussed in the TSO Command Processor PLMs.

TASK-CREATING COMMANDS

When the execution of a command includes the performance of a continuing system function (initiating or writing, for example), or when it involves the use of a serially reusable resource (such as the queue control records), the command is executed as a separate system task. Thus, the routine that issues the SVC 34 is not subject to an indefinite wait, and the function performed by the Command Execution routine is performed asynchronously with other tasks.

A Command Scheduling Control Block (CSCB) must be created for all task-creating commands. The CSCB Creation routine (module IGC0803D) is entered from IGC3703D when a CANCEL command is processed and the job to be cancelled is in the input or output queue; from IGC2203D when a VARY command has been processed with the CPU, channel, or storage operand; from the routing location module 1 (IGC7503D) for DISPLAY commands with the active, (C,K), consoles, jobname, M, N, PFK, Q, or U operand; and from IGC0403D for the other task-creating commands. (The CSCB Creation routine is also entered from module IGC5803D for the TSO commands, DISPLAY USER and SEND. For a detailed discussion of the processing of these commands, see the TSO Command Processor PLMs.)

The CSCB Creation routine examines the command to determine whether it is one of the following task-creating commands that are executed in the master scheduler region: DISPLAY A, DISPLAY C,K, DISPLAY M, DISPLAY PFK, DISPLAY U, QUIESCE, VARY CH, VARY CPU, and VARY STOR. If so, only one such command can be executed at a time, and the routine tests the common bit in the status flags field of the master scheduler resident data area. If the bit is on, it means that a command is being executed in the master scheduler region, and the CSCB Creation routine passes control to a message routine that informs the operator and returns control to the supervisor.

If the bit is off, the command can be executed; the CSCB Creation routine sets the bit on and constructs a CSCB for the command, then enqueues it in the chain and uses the POST macro instruction to notify the master scheduler.

If the command is not one that is executed in the master scheduler region, the CSCB Creation routine creates a CSCB and examines the verb code in the extended save area to determine whether the verb is START or MOUNT. If it is neither START nor MOUNT, the routine posts the master scheduler and returns control to the supervisor. If it is either START or MOUNT, the routine passes control to the START/MOUNT Hierarchy Parameter Processing routine (module IGC3803D). This routine processes the hierarchy parameter, posts the master scheduler, and passes control to the supervisor.

The master scheduler is the initial responder to all task-creating commands. It performs no analysis of the operand of the command however; it merely determines which task must be performed by analyzing the verb code in the CSCB, then attaches the task.

In response to the POST macro instruction issued in module IGC0803D (the CSCB Creation routine of the Command Scheduling routine), the Master Scheduler Wait and Attach routine (module IEEVWAIT) disables all interruptions and scans the chain of CSCBs looking for a CSCB in pending status. When it finds one, the master scheduler sets the CSCB assignment pending switch off, enables interruptions, and examines the verb code in the CSCB.

If the verb is START, the Master Scheduler Attach routine must assign a reader number, which is used by the interpreter as a part of the base for assigning unique data set names. The number is maintained in the master scheduler resident data area; when a task is started, the field is incremented by one, then stored in the CSCB.

If the command is not a START or MOUNT command, the command processing routine issues SVC 34 to have the Command Scheduling routine delete the CSCB from the chain (but not free the main storage it occupies) after issuing the ATTACH macro instruction.

The tasks of performing the DISPLAY A, DISPLAY U, DISPLAY CONSOLES, DISPLAY C,K, DISPLAY M, DISPLAY PFK, MONITOR A, DUMP, QUIESCE, VARY CH, VARY CPU, VARY STOR, and queue manipulation commands are attached with a protection key of zero, and in the supervisor mode; the tasks of performing the START and MOUNT commands are attached with a protection key of zero, but in the problem program mode.

Figure 30 lists the task-creating commands and indicates which execution routine the Attach routine calls to have the command processed.

In order to prevent the loss of any subsequent posts from the Command Scheduling routine, the Master Scheduler Attach routine repeats the scan, and attaches command execution tasks until there are no pending CSCBs in the chain. When this happens, control is returned to the Master Scheduler Wait routine, and the master scheduler goes into the wait state, until another POST macro instruction occurs.

The START Command

The operator issues a START command to initiate system tasks or user's job step tasks. System tasks are those privileged tasks, such as reading input streams and writing system output, that function with a protection key of zero and are listed by name in the linkage table IEEVLNKT (see Figure 34). Job step tasks function with a nonzero protection key and are not listed in the linkage table.

The description of a user's job has three sources: the procedure library, the input streams, and the operator. The procedure library contains job control language statements that describe frequently executed series of job steps. Job control language statements in an input stream may completely describe a job, or they may invoke a procedure, and modify it by overriding its statements or overriding fields within statements. The operator may furnish additional information in the operand of a START command; this information is used to invoke and modify a procedure.

The job description information for a system task, however, comes from three sources: the procedure library, JCL statements, and the operator. The procedure library contains standard descriptions of a reader, an initiator, and a writer (the user may add descriptions of other tasks to be established via START commands). JCL statements, (corresponding to input stream JCL) are stored internally; these statements invoke and modify the procedure. The operator furnishes additional information in the operand of the START command; this information is edited into the internally stored JCL statements before they are used to invoke and modify the procedure.

Command	First Module Executed	
	Load Module	Assembly Module
System Task Commands		
START	IEEPRWI2	IEEPRWI2
MOUNT	IEEPRWI2	IEEPRWI2
The DISPLAY A Command	IEEVDSP1	IEEVDSP1
The DISPLAY CONSOLES Command	IEEPDISC	IEEPDISC
The DUMP Command	IEE60110	IEE60110
The MONITOR A Command	IEEVDSP1	IEEVDSP1
Queue Manipulation Commands	IEEPALTR	IEEPALTR
CANCEL (job in queue)		
DISPLAY C,K		
DISPLAY jobname		
DISPLAY M		
DISPLAY N		
DISPLAY PFK		
DISPLAY Q		
DISPLAY U		
HOLD		
RELEASE		
RESET		
QUIESCE	IEEMPS03	IEEMPS03
VARY CH	IEEMPVCH	IEEMPVCH
VARY CPU	IEEMPVCP	IEEMPVCP
VARY STOR	IEEMPVSE	IEEMPVSE
TSO Operator Commands[1]		
DISPLAY USER	IEEVGPSD	IEEVGPSD
SEND	IEEVGPSD	IEEVGPSD

[1]See TSO Command Processor PLMs.

Figure 30. Task Creating Commands

Figure 31 summarizes the major attributes of and differences between system tasks, job step tasks initiated via a START command, and job step tasks initiated via the input stream.

The START command task is performed by the System Task Control routine (Figure 32); when the master scheduler determines that the command is START, it issues the ATTACH macro instruction to pass control to the Get Region routine (module IEEPRWI2), which is the first module of the System Task Control routine. The Get Region routine resides in the link pack area. It obtains a region from the specified hierarchy (the size is specified when the system is generated). IEEPRWI2 then establishes a STAE environment that contains exit and retry routines, to process abnormal-end-of-task conditions. Then the module passes control to the START Syntax Check routine (module IEEVSTAR), which is loaded into the new region.

If the system task control routine has an abnormal end of task condition, the STAE exit routine issues a message describing the condition to the operator. The STAE retry routine then attempts to de-allocate the existing device and releases the region. It issues an SVC 3 instruction to terminate the task.

	Protection key	Job step timing	May use SMF	May use checkpoint/ restart	Data set integrity	May be stopped, modified, cancelled	START CIB
System task	zero	no	no	no	yes except output writers	yes (note: cancellation may be done only during allocation)	yes
Job step task in a START command	nonzero	yes	no	no	yes	yes	yes
Job step task in input stream	nonzero	yes	yes	yes	yes	yes	no

Figure 31. Major Attributes of System Tasks, Jobstep Tasks in START Command, and Job Step Tasks in Input Stream

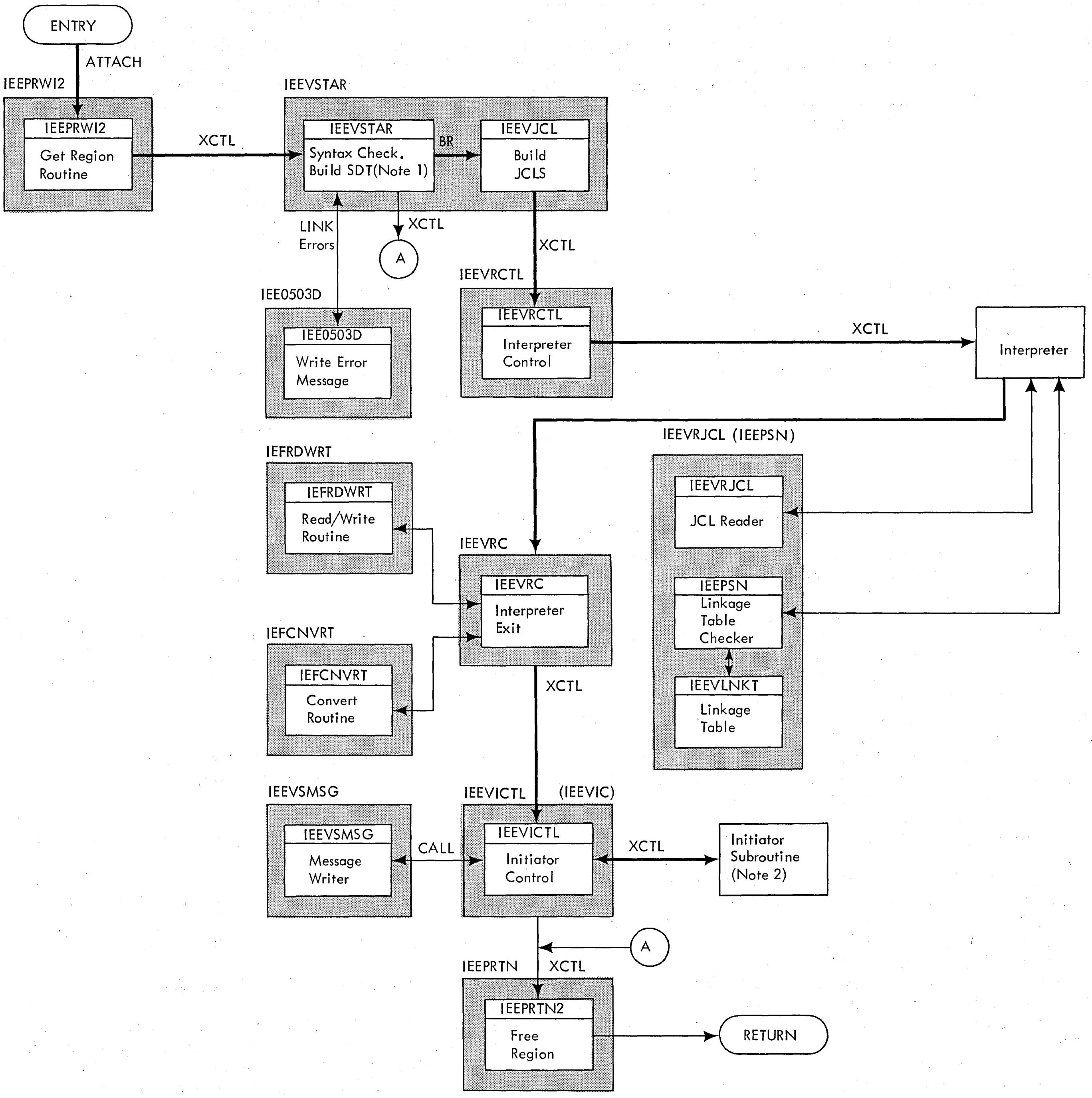
The Syntax Check routine gets main storage for and builds the start descriptor table (SDT) -- see Figure 33. JOB, EXEC, and DD statements are generated by the Syntax Check routine and placed in the SDT. Seven entries are provided in the SDT: the first entry contains the JOB statement, and the second entry contains the EXEC statement that calls the procedure specified in the START command; additional entries are provided for a DD statement and continuations of the EXEC and DD statements. The Syntax Check routine compares keyword parameters in the START command with a list of keyword parameters (but not DD subparameters) that are allowable in a DD statement or a JOB statement. If a keyword corresponds to a member of the list of DD parameters, the routine stores it in the DD statement in the SDT. This DD statement overrides the IEFRDER DD statement in the procedure specified in the START command. If a keyword does not appear in the list, it is assumed to be a symbolic parameter keyword and is placed in the EXEC statement in the SDT.

Finally, the Syntax Check routine passes control to the JCL Edit routine (module IEEVJCL), which builds the job control language set (JCLS). Using the information in the SDT, the JCL Edit routine puts the JCL in the form appropriate for the interpreter. Each statement is built in an 88-character buffer (obtained with a GETMAIN macro instruction). A pointer to the first buffer is placed in the CSCB associated with the START command. Each buffer contains a pointer to the next buffer, 4 bytes of reserved space, and a "card image" of the statement in the last 80 bytes.

After building the JCLSs, the JCL Edit routine builds the following:

- The Job Scheduling Entrance List (JSEL)
- The Job Scheduling Option List (JSOL)
- The Option Buffer (OPT)
- The Job Scheduling Exit List (JSXL).

The routine then releases the main storage occupied by the SDT and sets a flag in the JSXL to indicate a request for a post processing exit to entry point IEERGN in module IEEPRTNZ. The JCL Edit routine then issues an XCTL macro instruction to give control to the Job Scheduling subroutine at entry point IEEVRCTL of module IEEVRCTL.



Notes: General. The names in the upper left corners of the shaded blocks are load module names; the names in the unshaded blocks are assembly module (MODLIB) names.

1. There are two "Syntax Check" routines:
   - The START command uses load module IEEVSTAR, containing assembly modules IEEVSTAR and IEEVJCL.
   - The MOUNT command uses load module IEEVMNT1, containing assembly modules IEEVMNT1 and IEEVJCL.

2. The Initiator subroutine attaches module IEEVMNT2 if a MOUNT command is being processed.

Figure 32. The System Task Control Routine

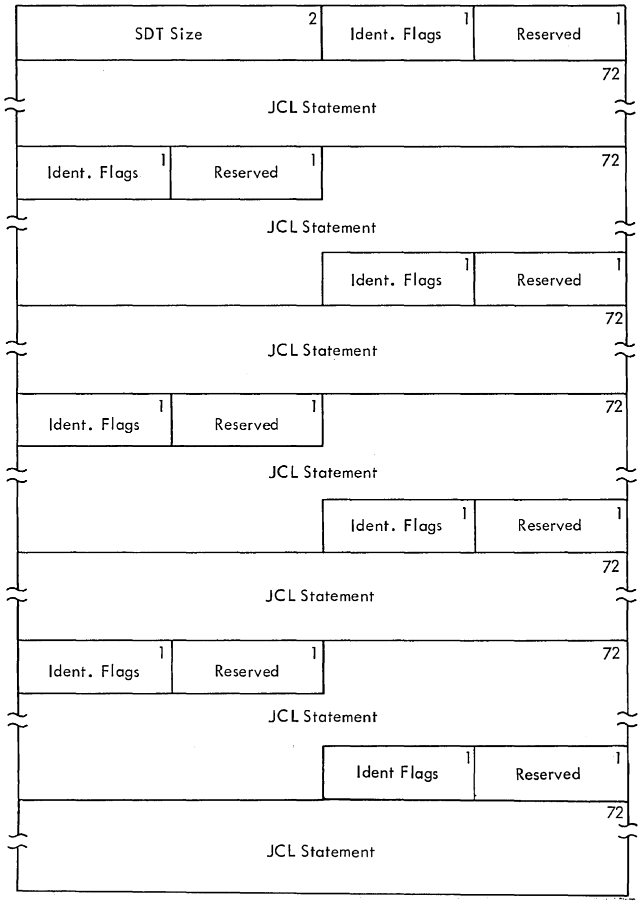


Figure 33. Start Descriptor Table (SDT)

Notes: The start descriptor table consists of up to seven entries, each of which contains a JCL statement (or a portion of a JCL statement) constructed by the Syntax Check routine of the System Task Control routine. In addition, each entry contains an identification flags field, whose bits, when set to one, have the following meanings:

Bit	Meaning
0	JOB statement
1	EXEC statement
2	DD statement
3	DD statement continuation
4	EXEC statement continuation
5-7	Reserved

The mapping macro instruction for the start descriptor table is IEESDT.

The Job Scheduling subroutine interprets the JCL for a single job, calls the initiator subroutine to initiate that job, and performs the necessary cleanup functions after regaining control from the Initiator subroutine. The information in the following lists determines the functions that the JSS and the Initiator subroutine will perform:

- The Job Scheduling Entrance List
- The Job Scheduling Options List
- The Job Scheduling Exit List
- The Option List (also known as the Option Buffer, or OPT) that the caller of the subroutine passes when giving control to the JSS.

A routine may call the JSS for any of the following reasons:

1. To complete the processing of a START command
2. To complete the processing of a MOUNT command
3. To complete the processing of a LOGON command that is entered from a TSO terminal.

The routine then uses the Q-manager routines IEFCNVRT and IEFRDWRT to write onto the job Q the LTH for the task to be executed. This permits normal termination processing of data sets to be performed at warm-start time.

The Reader/Interpreter (R/I) Control routine, IEEVRCTL, builds the interpreter entrance list (NEL), the interpreter option list, the interpreter exit list, and the job scheduling work area (JSWA). The third word of the NEL contains the address of the JCLS. Routine IEEVRCTL issues a LOAD macro instruction specifying the JCL reader routine IEEVRJCL, the interpreter post scan exit routine IEEPSN, and, when track stacking is specified, the load module IEFQMSK1, so that the reader may use these modules.

Control routine IEEVRCTL then creates entries in the NEL exit list for routines IEEVRJCL and IEEPSN, for IEFQMSK1 if track stacking is specified, for the SMF exit (routine IEFUJV) if specified, and for the interpreter exit routine IEEVRC. Next, routine IEEVRCTL processes the OPT options that the Job Scheduling subroutine has passed. If any options are present, the module gives control to (exits to) module IEFVSCAN, which processes the options and sets appropriate bits in the NEL and the JSWA.

Module IEFVSCAN verifies that the contents of the OPT conform to the syntax indicated by routine IEEVRCTL. IEFVSCAN scans the buffer from left to right (low address to high address). As each valid EBCDIC-encoded/operand in the buffer is encountered, the scan routine issues a CALL macro instruction for an exit routine defined for that operand. The exit routines (which are in module IEEVRCTL) transform the keyword options into bit settings in the NEL and the JSWA. Module IEFVSCAN then returns control to module IEEVRCTL and provides an indicative return code.

If module IEFVSCAN passes a return code other than zero, the caller (of IEEVRCTL) receives the code in the JSXL. Module IEEVRCTL then releases storage that had been obtained for the JSWA, JSOL, OPT, JSEL, NEL, and the exit list. Then, if so indicated in the JSXL, module IEEVRCTL takes a post processing exit. If no exit is available, IEEVRCTL returns control to the caller via the contents of register 14. The R/I control routine then frees the main storage used for the JSOL and the OPT, and passes control to the reader by means of an XCTL macro instruction.

The interpreter, used as a closed subroutine, is the same routine that performs the reading task. The non-zero value of the third word of the interpreter entrance list indicates that the input stream is an internal data set. Because the input stream is internal, the interpreter does a pseudo OPEN to bring a special access method into main storage, and stores a pointer to it in the input DCB. The special access method reads the records of the JCLS; it is entered from the expansion of the standard GET macro instruction.

The internally stored job control language statements, and the statements from the procedure library, are analyzed and combined normally. The standard job description tables are built, and an input queue entry is constructed, but because bit 7 of the option switches field of the option list is off, the entry is not enqueued, and the "job" cannot be selected by an initiator. If errors are detected during interpreter processing, the appropriate messages are placed in system message blocks, which are enqueued in the message class queue.

After it has scanned each JCL statement and stored it in the internal text buffer, the interpreter exits to the Interpreter Post Scan Exit routine (module IEEPSN), which is specified in the interpreter exit list. When the interpreter passes control to the Post Scan Exit routine, it passes a pointer to a parameter list, the first word of which is the address of the interpreter internal text buffer. The routine searches the buffer for an entry representing an EXEC statement that specifies a program name. Then the routine compares the program name specified in the EXEC statement with the list in the linkage table (see Figure 34) that contains all valid program names for procedures initiated by a START command.

If a valid program name is found in a multi-step job, the Interpreter Post Scan Exit routine sets an error return code in the communication area for the System Task Control routine.

If the program name is found in the linkage table, the routine checks another table that contains a list of all programs in the linkage table for which data set integrity is not provided. If the program name is found in the second table, an indicator is set in the communication area for the System Task Control routine.

After scanning the internal text buffer for each JCL statement, the routine returns control to the interpreter.

The interpreter places the main storage address of the job control table (JCT) in the NEL, and using the Interpreter Exit list established by routine IEEVRCTL, passes control via an XCTL macro instruction to the Interpreter Exit routine (module IEEVRC) with a code that indicates whether processing was successful.

The Interpreter Exit routine issues a DELETE macro instruction specifying, as operands, the following:

- The JCL reader routine
- The interpreter post scan exit routine
- The track stacking load module used by the interpreter.

In doing this, the routine uses the queue management routines IEFCNVRT and IEFDWRT to indicate that for a warm start, the job should be processed as if it had been dequeued from an input queue.

Then the routine analyzes the return codes from both the reader and the scan routines and performs the appropriate processing. If the codes indicate either JCL or I/O errors, the exit routine sets the appropriate error code in field CHSPA of the CSCB. (Appendix A explains the error codes that are set in the CSCB.) For all errors, routine IEEVRC sets the 'job-failed' bit in the JCT to request task/termination and passes the return codes (via the JSXL) to the caller.

The Interpreter Exit routine then frees main storage for the following:

- The JCLS
- The reader/interpreter register save area
- The NEL
- The OPT
- The exit list.

Then the routine issues an XCTL macro instruction to pass control to the Initiator Control routine (module IEEVICTL.)

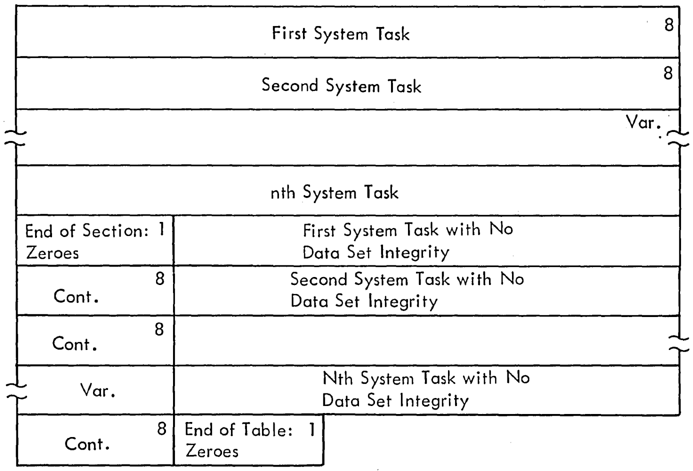


Figure 34. Linkage Table

Note: The linkage table contains all of the names that may be used in the PGM= name parameter of a procedure EXEC statement if the procedure is a system task.

The Initiator Control routine provides an interface between the System Task Control routine and the Intitator subroutine. This interface is similar to the one provided by the Interpreter Control routine between the System Task Control routine and the Interpreter subroutine. First the routine builds three parameter lists for the initiator: the initiator entrance list (IEL), the initiator option list, and the initiator exit list.

The option list, pointed to by the third word of the IEL, contains switches that indicate the processing that the initiator is to perform when invoked by the System Task Control routine. The Initiator Control routine sets the switches according to whether a system task or a problem program is being executed and/or according to information contained in the JSWA. The control routine then issues an XCTL macro instruction to pass control to the Initiator subroutine (at entry point IEFSD060). After performing the appropriate processing, the initiator returns control to entry point IEEVIC of the Initiator Control routine, as indicated in the initiator exit list.

A second function of the Initiator Control routine (other than to serve as an interface to the Initiator subroutine) is to handle errors found by the interpreter, the Interpreter Post Scan Exit routine, or the initiator. If errors are indicated, the routine issues a CALL macro instruction to pass control to the Message Writing routine (module IEEVSMSG), which writes the error message to the console.

Upon return from the initiator after termination of the task, the Initiator Control routine releases the main storage occupied by several control blocks built for the started task. Module IEEVICTL moves non-zero return codes from the IXL to the JSXL and indicates that the Initiator subroutine encountered the error. The module then processes the JSXL and takes an exit to entry point IEERGN of module IEEPRTNZ. The Free Region routine (module IEEPRTN2), which receives control from the Initiator Control routine via an XCTL macro instruction, is the last load module of the System Task Control routine. This routine, which resides in the link pack area, frees the CSCB, JSCB, and WTPCB if they exist, and then releases the region. It exits by branching to the master scheduler.

The MOUNT Command

The MOUNT command is used to reserve a unit for a particular volume. A unit will be allocated, a mount message issued, and the unit control block (UCB) will be marked; the system will not unload the volume until an UNLOAD command (see existing-task commands) is issued. The performance of the MOUNT command task, and the performance of the START command tasks by the System Task Control routine differ only in the module that checks syntax.

As is the case with the START commands, the information in the descriptor table is used to complete an internal data set that corresponds to JCL statements in the input stream. This data set is read, combined with statements from the MOUNT command procedure in the procedure library, and converted to tabular format by the interpreter. The tables are later used by the I/O Device Allocation routine, which is invoked by the Initiator subroutine.

As in the case of the START commands, the Job Scheduling subroutine is invoked. The JSS uses the reader/interpreter to interpret the JCLs and then passes control to the Initiator subroutine. The initiator, in turn, invokes the I/O Device Allocation routine and, upon return, attaches the Mount Command Bit Setting routine (module IEEVMNT2). This routine marks the UCB corresponding to the allocated unit "reserved" and either "public," "private," or "storage." It returns control to the Initiator subroutine.

The DISPLAY A Command

When the DISPLAY Commands Router routine (module IGC3503D) determines that a DISPLAY A Command is to be executed, it tests the region busy bit in the master scheduler resident data area. The DISPLAY A command is one of the commands that are executed in the master scheduler region; furthermore, only one DISPLAY A command can be executed for each call of the Display Active routine. Therefore, if the region busy bit is on, the command cannot be executed and the DISPLAY Commands Router routine passes control to a message module. The message module issues a message to the requesting operator and returns control to the supervisor.

If the region busy bit is not on, the DISPLAY Commands Router routine turns it on. Then the routine passes control to the Routing Location Module 1 (IGC7503D). This routine scans the command for routing (L=xxx) operands. If explicit routing operands have been specified, the routing routine places them in the extended save area (XSA). If no operands, or only partial operands, are specified, the routine checks the Routing Control Table for default routing operands. If it finds no defaults, the routine identifies the issuing console as the receiving console and defaults the area identifier to a null value (i.e., the first available area will be used). After the routine identifies the console address and the display area, it places them in the XSA. Then the routine passes control to the CSCB Creation routine (module IGC0803D) via an XCTL macro instruction. This routine creates and enqueues a CSCB for the command, posts the master scheduler, and returns control to the supervisor.

When a pending CSCB indicates that a DISPLAY A command is to be executed, the Master Scheduler Attach routine passes control to the Display Active routine via the ATTACH macro instruction.

The Display Active routine (module IEEVDSP1) disables the system for interruptions, then scans the supervisor TCB queue. When it finds a TCB containing a TIOT address, the routine determines the type of task, and performs the processing shown in Figure 35. If the task is the Master Scheduling task, a START-command task or system task with no subtasks, the Display Active routine passes control to the Display Region Size routine. Other command processing tasks (including the DISPLAY A task) are considered to be subtasks of the Master Scheduling task and are added to its subtask count. Subtasks of job steps and subtasks of system tasks (e.g., the System Output Writer's Writing subtask) are added to the appropriate subtask count.

The Display Region Size routine (module IEEVDRGN) is entered with a pointer to a TCB. The routine determines the boundaries of the hierarchy 0 and hierarchy 1 portions of the region assigned to the task and the amount of supervisor queue space used for the task, and returns the information to the Display Active routine.

When it reaches the end of the TCB queue, the Display Active routine extracts the job and step names from the TIOTs, enables the system for interruptions, and issues an SVC 110. Execution of SVC 110 causes control to be passed to the Master Scheduler Router routine (IGC00110), which passes control to the Active Task Message routine (IGC50110). The Active Task Message routine adds a control line and label lines to the display begun by IEEVDSP1. It uses the WTO macro instruction to pass the operator the following displays:

- Job and step names
- Subtask counts
- Region boundary addresses
- Supervisor queue space usage figures.

The Active Task Message routine then returns control to IEEVDSP1. This routine turns off the common bit and the display active flag, uses SVC 34 to unchain the CSCB (and free the storage it occupies), and returns control to the supervisor.

If the DISPLAY A command was issued in a TSO environment, the Display Active routine scans the region control block (RCB) table. If the command was issued from a terminal, the routine uses the TPUT macro instruction to pass information about the TSO region to the TSO operator. A detailed description of the TSO processing of the DISPLAY A command is in the TSO Command Processor PLMs.

The DISPLAY M Command

When the Display Router routine (IGC3503D) determines that a DISPLAY M command is to be executed, it passes control to the Routing Location Routine (IGC7503D), which scans the command for routing (L=xxx) operands. If explicit routing operands have been specified, module IGC7503D places them in the extended save area (XSA). If no operands, or only partial operands, are specified, the routine checks the routing control table for default routing operands.

TCBOTC Field	TCBLTC Field	TIOT	Task	See Note
Zero			Master Scheduling Task	3a
Master Scheduler TCB	Zero	Not Master Scheduler TIOT	START-Command Task	3a
Master Scheduler TCB	Nonzero	Not Master Scheduler TIOT	START-Command Task	3b
Master Scheduler TCB		Master Scheduler TIOT	Other Command Task	3c
START-Command TCB	Zero		System Task With No Subtasks	3a
START-Command TCB	Nonzero		System Task With No Subtasks	3b
Other TCB		Unique	Job Step With No Subtasks	3a
Other TCB		Not Unique	Job Step, Subtask of Job Step, or Subtask of System Task	3d

Notes:

1. The TCBOTC field points to the TCB of the task that established the current task.
2. The TCBLTC field points to the TCB of the first task established by the current task.
3. Depending upon the type of task, the processing performed is as follows:
   a. Enter the Display Region Size routine.
   b. Continue scan of supervisor TCB queue.
   c. Increment Master Scheduler Subtask count by 1, then continue scan of TCB queue.
   d. Increment origin task subtask count by 1, then continue scan of TCB queue.
4. If RJE, TCAM, or TSO is in the system, the search of the TCB by DISPLAY active is altered.

Figure 35. Task Identification in the Display Active Routine

If no default operands are found, the routine identifies the issuing console as the console to receive the display and defaults the display area identifier to a null value (i.e., the first available display area will be used). After the routine identifies the console address and the display area, it places them in the XSA. The Routing Location routine then checks the validity of the routing operands and, if no errors are found, passes control to the CSCB creation routine (IGC0803D). This routines enqueues a CSCB, posts the master scheduler, and returns control to the supervisor.

When a pending CSCB indicates that a DISPLAY M command is to be executed, the Resident WAIT/ATTACH routine (IEEVWAIT) passes control to the GET Region routine (IEEPALTR). This routine determines that a status display is to be created, and issues an SVC 110. The Master Scheduler Router, which is the first routine of SVC 110, determines that a DISPLAY M command is to be executed, and passes control to the Display Matrix Routines (IGC30110, IGC31110, and IGC32110). These routines build the display and pass it to the operator by using a WTO macro instruction.

The DISPLAY C,K Command

When the Verb Checker routine (IGC0403D) determines that a DISPLAY command has been entered, it passes control to the DISPLAY Command Router routine (IGC3503D). The router routine scans the command and, finding it to be a DISPLAY C,K command, passes control to the Routing Location routine (IGC7503D). The Routing Location routine determines the proper routing (L=xxx) operands and stores them in the XSA. It then passes control to the CSCB creation routine (IGC0803D), which enqueues a CSCB, posts the master scheduler, and returns control to the supervisor.

When a pending CSCB indicates that a DISPLAY C,K command is to be executed, the Resident WAIT/ATTACH routine (IEEVWAIT) passes control to the Get Region routine (IEEPALTR), which issues an SVC 110. The Master Scheduler Router, which is the first routine of SVC 110, determines the type of command entered and passes control to the

DISPLAY C,K Processor routines (IGC10110, IGC11110 and IGC12110). These routines build the display of CONTROL command operands and issue WTO macro instructions to pass the information to the operator.

The DISPLAY PFK Command

When the Display Router routine (IGC3503D) determines that a request for a display of the commands associated with a console's programmed function keyboard (PFK) key numbers has been entered, it passes control to the CSCB creation routine (IGC0803D). The CSCB Creation routine enqueues a CSCB for the command, posts the master scheduler, and returns control to the supervisor.

When a pending CSCB indicates that a DISPLAY PFK command is to be processed, the master scheduler WAIT/ATTACH routine (IEEVWAIT) passes control to the Get Region routine (IEEPALTR). This routine determines that a status display is to be created and issues an SVC 110 to continue processing of the request. The Master Scheduler Router routine (IGC00110), which is the first routine of SVC 110, determines that a DISPLAY PFK command is to be processed and passes control to the DISPLAY PFK routine (IGC40110). This routine builds the display and writes it to the operator's console by means of the WTO macro instruction. When the display is complete, control is returned to the supervisor.

The DISPLAY U Command

An operator issues the DISPLAY U command to request a tabular display of status information about specified units. When the Verb Checker routine (IGC0403D) determines that a DISPLAY U command has been entered, it passes control to the Display Router routine (IGC3503D). This routine passes control to the Routing Location routine (IGC7503D) which stores in the XSA the proper routing (L=xxx) operands for the command. Module IGC7503D passes control to the CSCB creation routine (IGC0803D) which enqueues a CSCB, posts the master scheduler, and returns control to the supervisor.

When a pending CSCB indicates that a DISPLAY U command is to be executed, the Resident WAIT/ATTACH routine (IEEVWAIT) passes control to the Get Region routine (IEEPALTR) which scans the command and issues an SVC 110 to pass control to the Master Scheduler Router routine (IGC00110). This routine determines that a DISPLAY U command is to be processed and passes control to the Unit Status routines (IGC20110, IGC21110, IGC22110 and IGC23110).

Upon entry, module IGC20110 issues a GETMAIN macro instruction to obtain a workarea. The routine then checks the command operands for correct syntax, and, if there are syntax errors, passes control to module IGC22110 to issue the appropriate error message and return control to the Master Scheduler Router. If there are no errors, module IGC20110 sets switches in the work area to indicate the units for which status has been requested, and passes control to module IGC21110. Module IGC21110 determines if the device name table is already in storage. If not, it issues a GETMAIN macro instruction to obtain an area into which the table can be loaded and passes control to module IGC23110. Module IGC23110 locates UCBs which satisfy the command specifications and builds a list of addresses in the workarea. It then passes control back to module IGC21110.

IGC21110 checks the workarea to determine which UCBs must be searched. It then assembles the display and places the information in the text line of the work area. The module then passes control to module IGC23110, which issues a WTO macro instruction to write each line of the display. When the display is complete, the module passes control to module IGC22110, which issues a FREEMAIN macro instruction to free the work area (and the device name table, if it was obtained by a GETMAIN macro instruction) and returns control to the Master Scheduler Router (IGC00110).

The DISPLAY CONSOLES Command

When the DISPLAY Commands Router routine (module IGC3503D) determines that the command to be executed is a DISPLAY CONSOLES command, it passes control to the Routine Location Module 1 (IGC7503D). This routine scans the command for routing (L=xxx) operands. If explicit routing operands have been specified, the routing routine places them in the extended save area (XSA). If no operands, or only partial operands, are specified, the routine checks the Routing Control Table for default routing operands. If it finds no defaults, the routine identifies the issuing console as the receiving console and defaults the area identifier to a null value (i.e., the first available area will be used). After the routine identifies the console address and the display area, it places them in the XSA. Then it passes control via an XCTL macro instruction to the CSCB Creation routine, which creates and enqueues a CSCB, posts the master scheduler, and returns control to the supervisor.

When a pending CSCB indicates that the command to be executed is a DISPLAY CONSOLES command, the Master Scheduler Attach routine uses the ATTACH macro instruction to pass control to the DISPLAY CONSOLES Get Region routine.

The DISPLAY CONSOLES Get routine (module IEEPDISC) resides in the link pack area. The routine obtains a region in which to execute the command, then uses the LINK macro instruction to pass control to the DISPLAY CONSOLES Processor routine (module IEEXEDNA).

The DISPLAY CONSOLES Processor routine (module IEEXEDNA) uses information from the unit control module (UCM) to construct messages that describe the console configuration.

When it is entered, the routine masks interruptions, then issues a header message. If hard copy is specified in the system, the routine prepares a descriptive message: if the device is a console, the information comes only from the UCM entry, but if the device is the system log device, information from the MCS prefix to the UCM base is also used.

The routine next constructs a message for each unit that was specified as a console when the system was generated.

When a message has been constructed, the DISPLAY CONSOLES Processor routine uses the WTO macro instruction to issue the message to the operator who entered the command. If the command was entered via an input stream, the message goes to the master console.

When all messages have been issued, the routine enables the system for interruptions and returns control to the DISPLAY CONSOLES Get Region routine (module IEEPDISC), which releases the region and returns control to the supervisor.

The MONITOR A Command

The MONITOR A command is one of the commands that is executed in the master scheduler region. When the MONITOR Commands Router routine (Module IGC7103D) determines that a MONITOR A command is to be executed, it tests the region busy bit in the master scheduler resident data area. If the region is busy (i.e., if the busy bit is on), the MONITOR Commands Router passes control to a message module. The message module issues a message to the requesting operator (indicating the region is busy) and returns control to the supervisor.

If the region busy bit is not on, the MONITOR Commands Router routine turns it on. Then the routine passes control to the Routing Location Routine (IGC7503D). This routine scans the command for the routing (L=xxx) operands. If explicit routing operands have been specified, the routine places them in the extended save area (XSA). If no operands, or only partial operands, are specified, the routine checks the Routing Control Table for default routing operands. If no defaults are found, the routine identifies the issuing console as the receiving console and defaults the display area identifier to a null value (the first available area will be used). When the console address and display area identifiers have been determined, the routine places them in the XSA.

The Routing Location routine then tests the validity of the routing operands, the routing authority of the issuing console, and the display-receiving capabilities of the designated console. If no incompatibilities are found, the routine tests to see if the requested console area is available. If the area is not available, the routine rejects the command and passes control to an appropriate error message module. If the area is available, the routine sets a MSGTYP=ACTIVE bit in the screen area control block (SACB) for the receiving console and the UCMMSG bit in the unit control module entry (UCME) of the receiving console. The routine then uses an XCTL macro instruction to pass control to the CSCB creation routine (module IGC0803D). This routine creates and enqueues a CSCB for the command, posts the master scheduler, and returns control to the supervisor.

When a pending CSCB indicates that a MONITOR A Command is to be initiated, the Master Scheduler WAIT/ATTACH routine determines if the system interval timer is operating. If it is not operating, the routine rejects the command and issues a message to the console. If it is operating, the routine establishes a time-interval that specifies an exit address in the Master Scheduler WAIT/ATTACH routine.

The routine at the exit address gets control initially and whenever the time-interval elapses. The routine checks a bit in the master scheduler region to see if a MONITOR A display is in progress. If the display is not in progress, the MONITOR A command processing was stopped during the time interval; the exit routine returns control to the supervisor. If MONITOR A command processing is in progress, the exit routine issues an STIMER macro instruction to re-establish the time interval and sets the CSCB pointer to zero. The routine then tests the region busy bit in the master scheduler region. If the bit is on, the region is in use, and the display cannot be updated; the routine passes an appropriate message to the operator and returns control to the supervisor. If the region is not busy, the Master Scheduler WAIT/ATTACH routine turns the busy bit on and uses the ATTACH macro instruction to pass control to the Display Active routine (IEEVDSP1).

The display active routine (IEEVDSP1) assembles the display by extracting information from the TCBs and by passing control to the display region size routine (IEEVDRGN) to determine the dimensions of the region assigned to each task. After the display active routine builds the display, it issues an SVC 110 to pass control to the Master Scheduler Router routine (IGC00110). This routine verifies that an active task (DISPLAY A or MONITOR A) command has been entered and passes control to the Active Task routine (IGC50110). The active task routine completes the building of the display and uses the WTO macro to pass the display to the operator. The routine then passes control back to the Display Active routine, which frees any workareas and returns control to the supervisor.

The DUMP Command

When an operator issues the DUMP command from a console, the router module (IEE0403D) identifies the command verb and issues an XCTL macro instruction to give control to the CSCB Create module (IEE0803D). Module IEE0803D inspects the operand field of the DUMP command. If COMM is either not specified or incorrectly specified, the module issues an XCTL macro instruction to give control to the message module (IEE7903D), which issues an invalid operand message. If COMM is specified, the create module establishes a command scheduling control block (CSCB) for the DUMP command and adds the CSCB to the CSCB chain for task-creating commands. This CSCB contains the verb code for DUMP and the operand field specified in the DUMP command. Then module IEE0803D issues a POST macro instruction to the master scheduling task.

In an MVT environment, the Wait and Attach routine (IEEVWAIT) checks the CSCB's assignment-pending bit and then issues an ATTACH macro instruction for the Queue Alter routine (IEEPALTR). The Alter routine identifies the command verb as DUMP and issues an SVC 110. (In an MFT environment, the Wait/Router routine (IEECIR51) checks the assignment-pending bit and issues the (SVC 110.)

The first load module (IEE00110) of the SVC 110 routine verifies that the command is DUMP and issues an XCTL macro instruction to give control to the Dump Command module (IEE60110). Module IEE60110 then issues a WTOR macro instruction requesting the operator to specify operands for the DUMP command. After the reply, module IEE60110 builds a parameter list based on the specified operand and issues an SVC DUMP (SVC 51). The SVC DUMP routine dumps the specified contents of main storage to a preallocated data set named SYS1.DUMP.

Figure 36 indicates the areas of main storage that are dumped according to the operand(s) specified in the DUMP command.

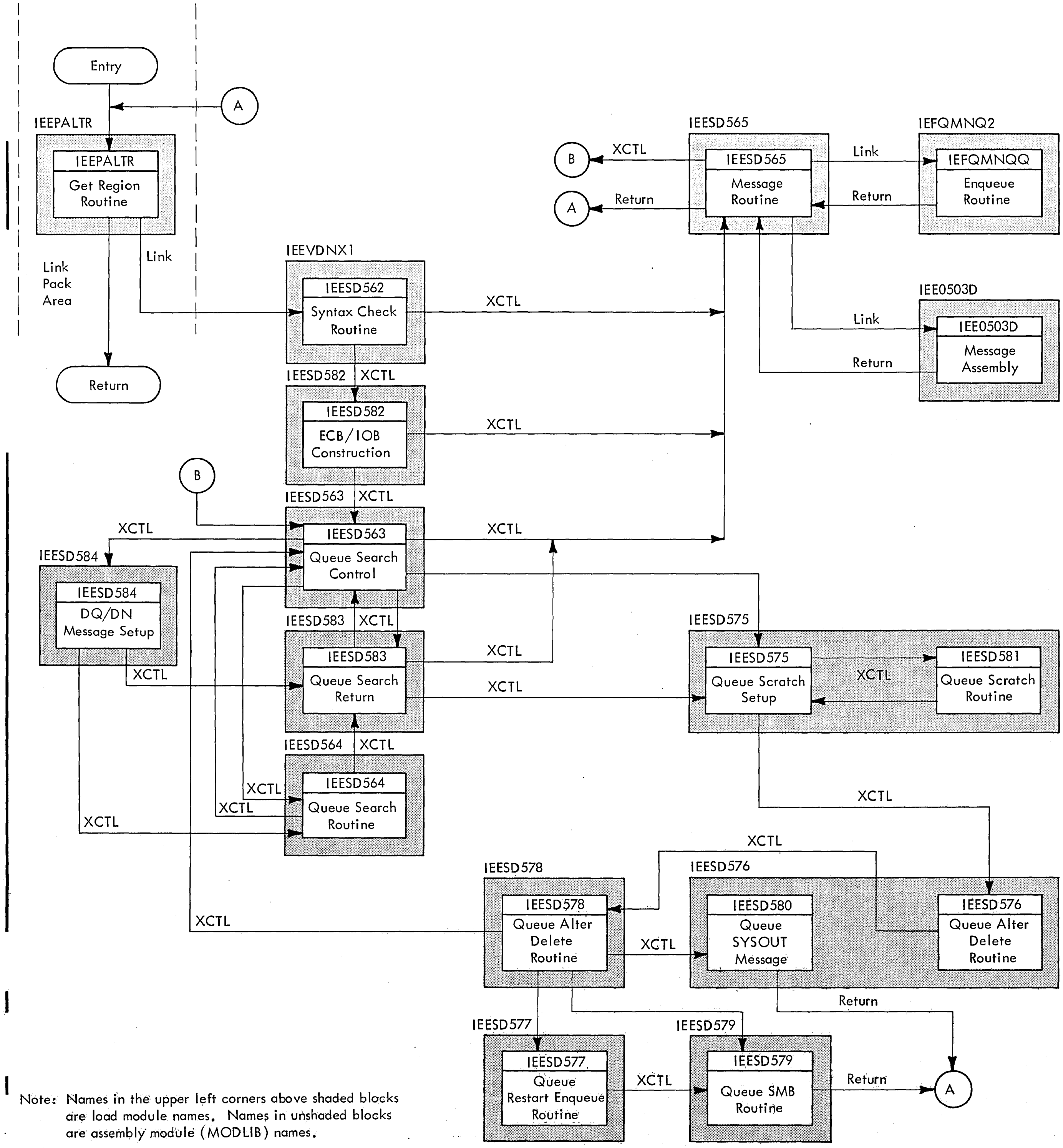
After the dump has been completed, the SVC DUMP routine returns control to the Dump Command module. The Dump Command module stores the return code (in register 15) in the extended save area (XSA) and issues an XCTL macro instruction for the Message module (IEE7903D). Module IEE7903D checks the return code and issues to the operator a message based on the return code.

Operand(s)	Main Storage Areas Dumped		
	Nucleus	System Queue Area (SQA)	Other
U(Default)	Yes*	Yes*	All
ALL	Yes*	Yes*	All
STOR...	No	No	Storage between specified addresses
STOR...,SDATA	Yes	Yes	Storage between specified addresses
SDATA	Yes	Yes	None
*The nucleus and the SQA areas are dumped before any other areas.			

Figure 36. DUMP Command Operand and Resulting Storage Dumped

The Queue Manipulation Commands

The execution of the CANCEL, DISPLAY N, DISPLAY Q, DISPLAY jobname, HOLD, RELEASE, and RESET commands requires access to or manipulation of the input, hold, and system output queues. The task of executing these commands is performed by the Queue Alter routine, which is described in Figure 37.



Note: Names in the upper left corners above shaded blocks are load module names. Names in unshaded blocks are assembly module (MODLIB) names.

Figure 37. The Queue Alter Routine

The Queue Alter routine, which is protected by a STAE environment (described in the next paragraph), is a collection of several modules that check command syntax, search and manipulate the queues, and issue messages to the operator. It is entered via an ATTACH macro instruction issued in the Master Scheduler Wait and Attach routine (module IEEVWAIT) after it has deleted the command CSCB from the chain.

The Get Region Routine (IEEPALTR)

The first module of the Queue Alter routine to be entered is the Get Region routine (module IEEPALTR). This routine, which resides in the link pack area, first determines whether the command entered is a DUMP command or a request for a status display (D C,K; D U; D M; OR D PFK). If so, the routine issues an SVC 110 to pass control to the Master Scheduler Router routine (IGC00110) for further processing of these commands (this processing is described under the various command titles in the COMMAND EXECUTION section). If the command entered is not a DUMP command or a request for a status display, the routine obtains a region of main storage for the task. The module then creates a STAE environment to provide for abnormal-end-of-task conditions. This environment contains exit and retry routines (within module IEEPALTR). Then IEEPALTR uses the LINK macro instruction to pass control to the Queue Alter Syntax Check routine (IEESD562).

When an abnormal end condition occurs, the STAE routine issues a message to the operator to inform him of the type of abnormal ending. Then the routine frees the region (including the subpools allocated within it) in main storage and returns control to the caller.

The Syntax Check Routine (Module IEESD562)

The Syntax Check routine issues the ENQ macro instruction to prevent access to the system work queues (SYS1.SYSJOBQE) by routines performing other tasks, then scans the operand of the command (which is stored in the CSCB) for valid syntax.

If the command syntax is in error, the Syntax Check routine passes control to the Service routine (module IEESD565), which uses the master scheduler Message Assembly routine (module IEE0503D) to issue a diagnostic message, and terminates the task. If the command syntax is valid, module IEFSD562 passes control to the ECB/IOB Construction routine (module IEESD582).

The ECB/IOB Construction Routine (Module IEESD582)

If the operand of the command contains a job name, module IEESD582 searches the chain of CSCBs for the corresponding CSCB. If a match is found and the command is a CANCEL command with no IN or OUT parameter specified, the routine issues a POST macro instruction specifying the ECB in the CSCB, then passes control to the Service routine. The Service routine (module IEESD565) uses the Message Assembly routine (module IEE0503D or module IEF2103D) to issue a message that informs the console or TSO terminal user that the job has been canceled. The Service routine then returns control to the caller. If the command is a CANCEL command with the IN or OUT parameter specified, or if the command is not a CANCEL command, the job has been selected for initiation and therefore the command cannot be processed; module IEESD582 passes control to the Service routine, which informs the operator and terminates the task.

If there is no CSCB for the job in the chain, a search of the system work queues is required to find the jobname. Module IEESD582 issues a GETMAIN macro instruction to obtain storage, and constructs an event control block (ECB) and an input/output block (IOB). Then, and if the operand of the command does not contain a jobname, module IEESD582 passes control to the Queue Search Control routine (module IEESD563).

The Queue Search Control Routine (Module IEESD563)

The Queue Search Control routine analyzes the codes set by the Syntax Check routine to determine what queue searching and manipulating functions need to be performed. The routine determines which queue is to be searched and reads the queue control record (QCR) for that queue.

- Upon initial entry, the Queue Search Control routine reads QCRs until it finds either

  a. the first non-empty queue, if the issued command is a DISPLAY Q or a DISPLAY N command, or
  b. a held queue, if the command is a DISPLAY Q command.

  It then passes control to the DISPLAY Q/DISPLAY N Message Setup Routine (module IEESD584).

- If all the queues selected for searching are empty, module IEESD563 passes control to the Queue Alter Service routine. This routine uses the master scheduler message assembly routine to issue a message to the operator.

- If a DISPLAY Q command was specified and an empty held queue has been found, the Queue Search Control routine passes control to the Queue Search Return routine (module IEESD583). Module IEESD583 adds the empty held queue to the queue display message that it issues.

- If the command is a HOLD Q command, the Queue Search Return routine reads the QCR for the specified queue, sets the hold queue bits in the QCR, then rewrites the QCR into SYS1.SYSJOBQE. If no queue is specified, the routine places all input queues in hold status. Finally, the routine passes control to the Service routine to issue a queue held message.

- If the command is a RELEASE Q command, the routine reads the QCR for the specified queue, turns off the hold queue bits in the QCR, rewrites the QCR into SYS1.SYSJOBQE, and posts the appropriate 'no-work' ECB in the ECB list. If no queue is specified, the routine releases all input queues and passes control to the Service routine to issue a queue released message.

- If a CANCEL command with the operand ALL or OUT has been specified, and if at least one jobname match was found during the search of all the output queues, module IEESD563 passes control to the Queue Scratch Setup routine (module IEESD575).

- For all other queue manipulation commands, the Queue Search Control routine establishes parameters for searching and passes control to the Queue Search routine (module IEESD564).

The Queue Search Return Routine (Module IEESD583)

After the Queue Search routine (module IEESD564) completes a queue search, it gives control to the Queue Search Return routine, which performs the following processing depending on the queue command and the results of the queue search:

- If the jobname match does not occur, the routine passes control to the Queue Alter Service routine, which uses the master scheduler Message Assembly routine to issue a message to the operator. Upon return, the Service routine returns control to the caller.

- If the jobname match occurs and the command is a CANCEL command, the routine passes control to the Queue Scratch Setup routine (module IEESD575).

- If the jobname match occurs and the command is a DISPLAY jobnames command, the routine issues a WTO or TPUT macro instruction to display job status information, then passes control to the Service routine.

- If the command is a DISPLAY Q command, the routine issues a WTO or TPUT macro instruction to display the number of entries in the queues that were searched, then passes control to the Service routine.

For the following command, no queue search is performed by module IEESD564:

- If the command is a DISPLAY Q command and an empty held queue is found, module IEESD583 receives control either directly from the Queue Search Control routine or on its first entry from the DISPLAY Q/DISPLAY N Message Setup routine. Module IEESD583 displays the hold status of the queue as part of the queue display message to the operator.

The Queue Search Routine (Module IEESD564)

The actual searching and manipulation of the queues is performed in the Queue Search routine (module IEESD564). Depending on the command and its operands, the following functions are performed:

- If the command is a CANCEL command, the routine searches the queues according to the parameters specified in the command. If no parameters are indicated, the input, background reader, and hold queues are searched for the job's queue entry. If IN is specified, the input and background reader queues are searched; the search ends if the entry is found. If a specific input queue is specified, only that queue is searched. If OUT is specified, all output queues are searched unless a specific queue was indicated. If ALL is specified, the input, background reader, and hold queues are searched first. If the job's entry is found, the output queues are not searched; otherwise, the output queues are searched, too. Whenever a job's queue entry is found, the entry is dequeued.

- If the command is a DISPLAY jobname command, the routine searches the input, hold, background reader, ASB, and output queues. When it finds the job's entry, the routine returns job status information to the Queue Search Return routine, which then returns the job's status to the operator.

- If the command is a DISPLAY N command, the routine searches the queues specified in the command (if no specification is made, all queues are searched). The Queue Search routine issues a WTO or TPUT macro instruction indicating the name of each job, and the queue in which it is found.

- If the command is a DISPLAY Q command, the routine searches the queues specified in the command (if no specification is made, all queues are searched). It returns the number of entries in each queue searched to the Queue Search Return routine.

- If the command is a HOLD jobname command, the routine searches the input queues (and if necessary the hold queue) for the job's entry. If the entry is found in an input queue, the routine dequeues it, and stores the name of the input queue in the entry; the Service routine uses the Queue Management Enqueue routine to place the entry in the hold queue. If the entry is found in the hold queue, or if the job is not found, the Service routine informs the operator.

- If the command is a RELEASE jobname command, the routine dequeues the entry from the hold queue. The Service routine uses the Queue Management Enqueue routine to reenqueue the entry in its original input queue.

- If the command is a RESET command, the Service routine performs the specified changes in class and priority. This is accomplished by dequeueing the job's entry and using the Queue Management Enqueue routine to reenqueue the entry in the specified queue, at the specified priority. All queues are searched, starting with the input queues and the hold queue. A change in class for an entry found in the input queue causes the entry to be placed in the queue of the new class. A class change for a hold queue entry is placed in the function code ID field of the entry's logical track header record. If the job's entry is not found in the input queues, the background reader queue, or in the hold queue, and a priority change is called for, the priorities of all output entries for the job are reset. If the job is being processed when the command is issued, the routine stores the new priority in the input queue QMPA in the job's CSCB. When the job terminates, the Termination routine uses that priority to enqueue the output queue entries.

  If the output queue qualifier is specified, only the specified output queue is searched and only one entry is changed in class and priority. The entry that is changed is the first entry of the highest priority for the specified job. If the job is being processed when the command is issued or if no entry for the job is found, the operator is informed and no action is taken.

The DISPLAY Q/DISPLAY N Message Setup Routine (Module IEESD584)

The DISPLAY Q/DISPLAY N Message Setup routine receives control only once from the Queue Search Control, either:

- when the first non-empty queue is found during processing for either a DISPLAY Q or a DISPLAY N command, or

- when a held queue is found during processing for a DISPLAY Q command.

The routine issues the WTO or the TPUT macro instruction to display the DISPLAY Q/DISPLAY N control and label lines. If entry to the routine was due to a held queue being found, module IEESD584 passes control to the Queue Search Return routine (module IEESD583). This routine adds this queue to the display message it issues to the operator. Otherwise, module IEESD584 passes control to the Queue Search routine (module IEESD564) to begin the queue search.

The Queue Alter Service Routine (Module IEESD565)

The Queue Alter Service routine issues a DEQ macro instruction to allow routines performing other tasks access to the QCRs. Based on information passed by the calling routine, the Service routine performs the following functions:

- If a queue entry is to be reenqueued, it passes control via a LINK macro instruction to the Queue Management Enqueue routine (module IEFQMNQQ).

- If a message is to be written to the operator, it passes control via a LINK macro instruction to the master scheduler Message Assembly routine (module IEF0503D).

- If additional system output queues must be searched, the routine uses an XCTL macro instruction to pass control to the Queue Search Control routine (module IEESD563).

- It issues a FREEMAIN macro instruction to free the ECB/IOB that was used to read SYS1.SYSJOBQE.

After the requested processing is completed, the Service routine returns control to the Get Region routine (module IEEPALTR), which releases the region of main storage obtained for the task and returns control to the supervisor.

The Queue Scratch Setup Routine (Module IEESD575)

The Queue Scratch Setup routine builds the parameter list for the SCRATCH macro instruction (SVC 29) according to whether the cancelled job is in the input or output queues. If the job is in the input queue, the Queue Scratch Setup routine determines whether there are SYSIN data sets to be scratched. If so, the routine passes control to the Queue Scratch routine (module IEESD581). After the Queue Scratch routine has scratched all data sets, or if there were no data sets to be scratched, the Queue Scratch Setup routine passes control to the Queue Alter Delete routine.

The Queue Alter Delete Routine (Module IEESD576)

The Queue Alter Delete routine passes control to the Queue Management Delete routine (module IEFQDELE) to delete the queue entries associated with the cancelled job. Upon return from the Queue Management Delete routine, this routine passes control to the Queue Message Class Setup routine (module IEESD578).

The Queue Restart Enqueue Routine (Module IEESD577)

The Queue Restart Enqueue routine passes control to the Queue Management Enqueue routine (module IEFQMNQQ) to enqueue SYSOUT data sets for cancelled restarting jobs. Upon return from module IEFQMNQQ, the Queue Restart Enqueue routine passes control to the Queue SMB routine (module IEFSD579).

The Queue Message Class Setup Routine (Module IEESD578)

The Queue Message Class Setup routine places zeros in the DSBs for the message class and sets up the QMPA for enqueuing the message class. If the job is a restarting job, the routine passes control to the Queue Restart Enqueue routine. If the job is in the output queue, with more queues to be searched, the routine passes control to the Queue Search Control routine (module IEESD563). If the CANCEL command is issued for a class other than the message class or for a job that has no message class, the routine passes control to the Message routine (module IEESD580). Otherwise, it passes control to the Queue SMB routine (module IEESD579).

The Queue SMB Routine (Module IEESD579)

The Queue SMB routine places the appropriate cancel message into the first SMB and passes control to the Queue Management Enqueue routine (module IEFQMNQQ) to enqueue the message class. The Queue SMB routine also issues the cancel message to the operator and returns control to the Get Region routine (module IEEPALTR).

The Message Routine (Module IEESD580)

The Message routine issues the cancel message to the operator if there is no message class associated with the job. The routine then returns control to the Get Region routine (module IEEPALTR).

The Queue Scratch Routine (Module IEESD581)

The Queue Scratch routine issues the SCRATCH macro instruction (SVC 29). Upon return from the SVC routine, the Queue Scratch routine issues a "data set not deleted" message if the return code is nonzero. The routine returns control to the Queue Scratch Setup routine (module IEESD575).

The System Reconfiguration Commands

System reconfiguration is the process of changing, physically or logically, the type or quantity of elements available to the system. Physical reconfiguration connects or disconnects elements from the system. Logical reconfiguration, which is accomplished by programming, changes system tables to notify the control program of the number and types of elements available to it. Logical reconfiguration may be performed without performing physical reconfiguration, but should always be done when physical reconfiguration takes place.

Physical reconfiguration can be performed when the system is operating if the QUIESCE command is issued before the reconfiguration takes place. The QUIESCE command, which is valid only in systems that include the multiprocessing option, suspends system activity until the operator signals that the system may continue.

Logical reconfiguration may be performed when the system is loaded (see the IPL/NIP Program Logic Manual). It may also be performed as a result of issuing the VARY command, which changes the status of paths, devices, channels, CPUs, or areas of main storage to online or offline. The VARY Channel, VARY CPU, and VARY Storage commands (which are valid only in systems that include MVT with Model 65 Multiprocessing) are discussed in this section, immediately following the discussion of the QUIESCE command.

The QUIESCE Command

Before a physical reconfiguration can take place, system activity must be suspended. In order to suspend system activity, the operator issues the QUIESCE command, which is identified in the Command Scheduling routine. The Command Scheduling routine posts the Master Scheduler, and the Master Scheduler Attach routine uses the ATTACH macro instruction to pass control to the Quiesce routine.

When it is entered, the Quiesce routine (module IEEMPS03) disables the system for all interruptions. If there are two CPUs in the system, this locks out the other CPU (prevents it from executing instructions in the Supervisor routines). The Quiesce routine then marks all TCBs (except its own) non-dispatchable and uses the Task Removal routine to issue a shoulder tap to the other CPU. This forces the other CPU to halt execution of its currently active task, and the Quiesce task becomes the only task in the system that can be executed. Thus, no further I/O operations will be initiated, although there may be I/O operations in progress. These I/O operations must be allowed to complete before the system can be stopped and the reconfiguration performed.

The Quiesce routine scans the system UCBs, looking for any that indicate an incomplete I/O operation. If it finds that there are I/O operations in progress, the Quiesce routine enables the system for interruptions, sets the timer for a ten-second interval, and waits for the interval to expire.

When the timer interval has expired, the Quiesce routine disables the system for interruptions and scans the UCBs. If there are still I/O operations in progress, the routine again enables the system for interruptions and requests a ten-second timer interval.

If, after 24 ten-second intervals have been used, an I/O operation is still in progress, the Quiesce routine assumes that the QUIESCE command cannot be executed (because a channel program is in a loop, for example). In this case, the routine sets all TCBs dispatchable and sets the IEATCBP field to zero in each prefixed storage area (PSA). This causes both CPUs to start dispatching at the top of the ready queue. Finally, the routine uses the WTO macro instruction to inform the operator that the command could not be executed, and returns control to the supervisor.

If, however, the Quiesce routine finds that all I/O operations have been completed, it prepares the CPUs for stopping:

- It saves the contents of the active timer, then sets the active timer to a large value.
- It uses a shoulder tap to cause the other CPU to execute module IEEMPS00. This routine loads a PSW disabled for all interruptions, sets up the External, Machine Check, and I/O New PSWs to trap interruptions (because bringing power up or down may cause false interruption signals to be generated), and executes the Diagnose Stop instruction.
- It performs the same operations performed in module IEEMPS00, but in its own CPU.

The Diagnose Stop instruction causes a CPU to enter the stopped state, and the Manual light to be lit. When the Manual lights are lit on both CPUs, the operator may perform the physical reconfiguration procedures.

When the physical reconfiguration procedures have been completed, the operator depresses the Start key on each CPU. Each CPU then clears any pending machine checks, sets the CPUWAIT bit in its prefixed storage area (PSA), and tests the other CPU's CPUWAIT bit until it has been set.

When both CPUs have cleared all pending machine checks, they clear any pending external or I/O interruptions. Then, while the other CPU loads a PSW to put it in the wait state and enable it for interruptions, the CPU that is executing the QUIESCE command restores the active timer, sets the PSA field IEATCBP to zero for both CPUs, turns off the common bit in the status flags field of the master scheduler resident data area, and returns control to the supervisor.

The VARY Commands

The VARY command verb causes the Router routine of the Command Scheduling routine to pass control to the VARY Keyword Router routine (module IGC3203D). This routine identifies the first keyword in the command and passes control to the appropriate keyword processing routine. If the command is one of the three task-creating VARY commands (VARY CH, VARY CPU, and VARY STOR), and if MCS is in the system, the VARY Keyword Router routine passes control to the MCS VARY Syntax Check routine (module IGC3303D). If MCS is not included, the VARY Keyword Router routine passes control to the VARY/UNLOAD Syntax Scan routine (module IGC1103D). In either case, a unit scan is performed, and control is passed to the VARY Operand Check routine (module IGC2203D). The VARY Operand Check routine performs a syntax check of the command operand. If errors are found, the routine passes control to a message module which informs the operator and returns control to the supervisor. If no errors are found, it passes control to the CSCB Creation routine (module IGC0803D). The CSCB Creation routine creates and enqueues a CSCB for the command, posts the master scheduler, and returns control to the supervisor.

The Master Scheduler Attach routine uses the ATTACH macro instruction to pass control to the appropriate module:

Command	Module
VARY CH	IEEMPVCH
VARY CPU	IEEMPVCP
VARY STOR	IEEMPVSE

The VARY Channel Command

The operator issues a VARY channel command to change the status of a channel to online or offline.

When it is entered, the VARY Channel routine (module IEEMPVCH) makes several decisions, and on the basis of these decisions, it executes the command, rejects it, or informs the operator that the channel is already in the requested status:

- It checks the command operands in the CSCB to ensure that valid operands were used.

- It checks the CSCB to determine whether the option is ONLINE or OFFLINE.

- It checks the channel availability table in the PSA to determine whether the channel is actually in the system, whether it is online or offline, and whether it is operational.

- For OFFLINE requests, the routine checks to ensure that the reserved path to a shared direct access storage device will not be made unavailable.

- For OFFLINE requests, if the channel represents the last path to a device (multiprocessing supports two paths to each device from each CPU), the routine checks the command operands in the CSCB for "UNCOND". The "UNCOND" operand is effective only if the path is not to an allocated device, to a permanently resident data set, or to a tape device.

- It checks to ensure that executing the command will not remove the last path to an operator's console.

These checks, and the actions performed as a result, are shown in Figure 38, the VARY Channel Decision Table.

Online: The VARY Channel routine determines if the channel to be placed online is already online; if it is, the WTO "Channel Online" message is issued. If the channel is offline, the VARY Channel routine marks all TCBs, except its own, nondispatchable. It then uses the Task Removal routine to force the other CPU to halt execution of its currently active task.

The online routine must be executed on the CPU specified in the command. To get onto the proper CPU, the routine places the active timer on the desired CPU and issues an STIMER wait. When the timer interruption occurs, the VARY Channel task is dispatched on the CPU that has the active timer. For the duration of the STIMER wait, external interruptions are enabled.

When the interval expires, the routine marks the channel online in the channel availability table, and uses the Multiprocessing Test Channel routine in the I/O supervisor to determine whether the channel to be brought online is operational, and whether there are any catastrophic channel errors. The routine also uses the MAP function of the IOSGEN macro to determine which paths to the devices attached to this channel are available.

If for any reason the channel cannot be used, the routine marks the channel offline and uses the WTO macro instruction to inform the operator that the channel is offline. Next the routine determines whether MCS is in the system. If it is not a new alternate console may be designated by the system. The routine then marks all

Operands Not Valid	X										
Operands Valid		X	X	X	X	X	X	X	X	X	X
ONLINE Option		X	X	X							
OFFLINE Option					X	X	X	X	X	X	X
Channel Online		X				X	X	X	X	X	X
Channel Offline			X	X	X						
Channel Operational			X								
Channel Not Operational				X							
No UNCOND Operand							X	X			
UNCOND Operand									X	X	
Channel Represents No Last Paths to Any Devices						X					
Channel Represents Last Path to an Allocated Device, a Permanently Resident Data Set, or a Tape Device							X		X		
Channel Represents Last Path That is Not to an Allocated Device, a Permanently Resident Data Set, a Permanently Mounted Device, or a Tape Device								X		X	
Channel Represents Last Path to Operator's Console											X
Do Action Number	1	2	3	1	5	4	1	1	1	4	1

Actions:
1. Reject Command
2. WTO "Channel Online" message
3. Execute VARY Channel ONLINE
4. Execute VARY Channel OFFLINE
5. WTO "Channel Offline" message

Figure 38. VARY Channel Decision Table

tasks dispatchable, issues a dummy shoulder tap to restart the channel being varied online, unlocks the supervisor, and returns control to the supervisor.

Offline: The VARY Channel routine determines if the channel to be placed offline is already offline; if it is, the WTO "Channel Offline" message is issued. If the channel is online, the routine uses the MAP function of the IOSGEN macro to determine whether the channel to be varied represents the last path to one of its devices. If so, the CSCB is checked for the "UNCOND" operand. If "UNCOND" was not specified, control is returned to the supervisor. If "UNCOND" was specified, the routine determines if the device is allocated or permanently mounted or contains a permanently resident data set.

If any of these conditions exist, the command is rejected and control is returned to the supervisor. If none of these conditions exist, the device is checked to ensure that it is not the only remaining operator's console. If it is the only console, the command is rejected and control is returned to the supervisor. If it is not the only console, the channel is eligible to be marked offline.

Next the VARY Channel routine marks the channel offline in the channel availability table, uses the WTO macro instruction to inform the operator that the command has been executed, turns off the common bit in the status field of the master scheduler resident data area, and returns control to the supervisor.

If MCS is in the system, and placing a channel offline will result in removing the last path to any console, the VARY Channel routine issues a message to the master console operator advising him that his command is rejected and providing him with the unit addresses of all active asymmetric consoles on the channel.

If MCS is not in the system, and a channel is to be placed offline, the VARY Channel routine examines the UCM entries corresponding to the units on the channel to determine whether the channel represents the last path to any console. If so, and if the console affected is the primary console, a swap is performed so that the alternate console becomes the primary console, and a new alternate is found from the list of consoles in the UCM. If the alternate console is the console affected, a new alternate is found.

The VARY CPU Command

The operator issues a VARY CPU command to change the status of a CPU to online or offline.

When it is entered, the VARY CPU routine (module IEEMPVCP) first enqueues the UCBs and then makes several decisions. On the basis of these decisions, it executes the command, rejects it, or informs the operator that the specified CPU is already in the requested state:

- It checks the command operands in the CSCB to ensure that valid operands were used.

- It checks the command operands in the CSCB to determine whether the option is ONLINE or OFFLINE.

- It checks the CPUSTAT byte in the PSA to determine whether the system is in multiprocessing or partitioned mode, whether one or two CPUs are online, and (if one CPU is online) which CPU is online.

- For OFFLINE requests, the VARY CPU routine checks to ensure that the CPU being varied offline is not an element of the reserved path to a shared direct access storage device.

- For OFFLINE requests, if the CPU represents the last path to a device (multiprocessing supports two paths to each device from each CPU), the routine checks the command operands in the CSCB for "UNCOND". The "UNCOND" operand is effective only if the path is <u>not</u> to an allocated device, to a permanently resident data set, or to a tape device.

These checks, and the actions performed as a result are shown in Figure 39, the VARY CPU Decision Table.

<u>Online</u>: The VARY CPU routine determines if the CPU to be varied online is already online; if it is, the WTO "CPU Online" message is issued. If the CPU to be varied online is offline, the VARY CPU routine loads location zero of each PSA with a PSW. The PSW that is stored in the PSA assigned to the object CPU points to the initialization routine to be executed by that CPU. The PSW that is stored in the other PSA points to an error routine that will be executed by the object CPU if the operator has the two prefix switches set symmetrically.

When it has stored the PSWs, the VARY CPU routine executes an External Start instruction and begins testing the VRYCPU byte located in the VARY CPU routine. This byte is set to X'02' by the CPU to be varied at the beginning of its initialization routine. The VARY CPU routine tests for X'02'; if the test is not successful after X'2000X' tries, the routine assumes that the External Start instruction was not successful. In that case, the routine uses the WTO macro instruction to inform the operator that the command could not be executed and returns control to the supervisor. If, however, the test is successful, the routine switches the CPU IDs in the lock byte, informs the CPU to be varied that initialization is complete, and enters an enabled wait state.

Operands Not Valid	X												
Operands Valid		X	X	X	X	X	X	X	X	X	X	X	X
ONLINE Option		X	X	X	X								
OFFLINE Option						X	X	X	X	X	X	X	X
Multiprocessing Mode			X	X	X		X	X	X	X	X	X	X
Partitioned Mode		X				X							
1 CPU Online			X		X		X	X					
2 CPUs Online				X					X	X	X	X	X
CPU (to be Varied) is online			X	X			X		X	X	X	X	X
CPU (to be Varied) is offline					X			X					
No UNCOND operand										X	X		
UNCOND operand												X	X
CPU Represents No Last Path to Any Devices									X				
Channel on CPU Represents Last Path to an Allocated Device, a Permanently Resident Data Set, or a Tape Device										X		X	
Channel on CPU Represents Last Path that is Not to an Allocated Device a Permanently Resident Data Set, or a tape device											X		X
Do Action Number	1	1	4	4	5	1	1	2	3	1	1	1	3

Action:
1. Reject Command
2. WTO "CPU Offline" Message
3. Execute VARY CPU OFFLINE
4. WTO "CPU Online" Message
5. Execute VARY CPU ONLINE

Figure 39. VARY CPU Decision Table

When it is entered, the initialization routine for the CPU to be varied clears its own registers and determines whether the timer is enabled. If the timer is not enabled, the initialization routine sets a flag so that the Vary CPU routine will issue a warning message that the timer is not enabled.

Next, the initialization routine saves its external new and I/O new PSWs. In order to trap any extraneous interruption signals that may have been generated, the routine loads a PSW that is enabled for interruptions, then loads a disabled PSW. It initializes the external and machine check new PSWs for multiprocessing, initializes its PSA, and then takes the following steps to verify that the channels, control units, devices, and paths assigned to the CPU to be varied are indeed available to it:

- It sets bit three in each channel availability table entry to zero. This bit is used only by the VARY CPU routine to indicate that the channel status is a result of a VARY CPU command. When the CPU is again placed online, the channel should therefore be available to it. And, since this bit is independent of VARY Channel, any channels varied offline by VARY Channel remain offline.

- The routine uses the I/O Supervisor Test Channel routine to ensure that all channels not marked offline in the channel availability table are operational. If a channel is not operational, the initialization routine sets bit one in the channel availability table entry to mark the channel offline; when the VARY CPU routine informs the operator that the command has been executed, it also notifies the operator of any channels on the CPU that are marked offline.

- The routine uses the MAP function of the IOSGEN macro to get the path indicators for each UCB.

- The routine uses the I/O Supervisor Test I/O routine to test the online paths from this CPU to all devices except offline direct access devices. If the return code indicates that the unit is available, the UCB is marked online and tape units are marked ready. A unit check from any device except tape and direct access is treated as a good path and the device is marked online. If the return code indicates that the paths are unavailable, the routine uses the IOSGEN macro instruction to mark the paths offline if there is another path available to the device and the UCB is marked online; otherwise, the path remains marked online.

The routine initializes the CONSOLID field in the object CPU's PSA with the address of its operator console if it is operational. Then, it determines if MCS is in the system.

If not a new alternate console may be designated by the system.

When channel and unit availability have been verified, the initialization routine initializes the CPUSTAT byte in both PSAs to X'00', sets the second word of both IEATCBP fields to zero, and initializes its External and Machine Check New PSWs for multiprocessing. A dummy shoulder tap is initiated to restart the channels on the CPU just brought online. For systems generated with the system management facilities, a record containing the CPU ID, the time, date, and the online routine is built and written. All tasks are set dispatchable.

The routine uses the WTO macro instruction to inform the operator that the command has been executed and to notify him of any channels that are offline. The routine dequeues on the UCBs, turns off the common bit in the status flags field of the master scheduler resident data area, and returns control to the supervisor.

Offline: The VARY CPU routine determines if the CPU to be placed offline is already offline; if it is, the WTO "CPU Offline" message is issued. If the CPU is online, the VARY CPU routine determines whether the active timer and (in systems without MCS) console device are on the CPU to be varied offline or on the CPU that will remain online. The routine inspects the PREFTMRA field in the PSA for each CPU; the field in the PSA for the CPU containing the active timer is set at zero, and if that CPU is to be placed offline, the routine switches the active timer to the other CPU by exchanging the contents of the two PREFTMRA and timer fields. If (in systems without MCS) the channel containing the active console device is on the CPU to be varied (i.e., the object CPU), the routine posts the console ECB, causing the Console Communications routine to switch to the alternate console device.

In system with MCS, the VARY CPU routine examines the UCM entries to determine whether there are any active asymmetric consoles on the CPU. If so, the routine issues a message to the master console operator advising him that his command is rejected, and providing him with the unit addresses of all active asymmetric consoles on that CPU.

Using the MAP function of the IOSGEN macro of the I/O Supervisor, the routine checks the paths of each device attached to a channel that is online on the object CPU to determine if the last available logical path to the device is on the object CPU. The command is rejected if the last path is on the object CPU unless the "UNCOND" parameter has been specified and the last path is to a device that is not allocated, does not contain any permanently resident data sets, and is not a tape device. Any device that will have its last path removed is marked offline by the pending bit in the UCB and the last path bit in the negative UCB. The operator's console on the object CPU is always taken offline, regardless of the conditional parameter.

When the active timer and (in systems without MCS) console device have been switched to the CPU that is to remain online, the routine sets all TCBs except its own non-dispatchable, and uses the Task Removal routine to force the other CPU to halt execution of its currently active task.

The VARY CPU routine then searches the UCBs of the object CPU to determine whether all I/O operations on that CPU have been completed.

If there are I/O operations in progress, the routine sets the timer for a five-second interval and waits for the interval to expire. When the interval has expired, the routine again tests the UCBs for incomplete I/O operations; the routine continues requesting five-second timer intervals and testing UCBs until all I/O operations are completed, or until three minutes have elapsed. The command is rejected after three minutes if all of the I/O has not been completed.

When all of the I/O operations have been completed, the Vary CPU routine marks the channels offline on the object CPU offline. If MCS is not in the system, the alternate console routine determines if a new alternate must be found and searches the UCMs to find another alternate if the current one is scheduled to go offline. The VARY CPU routine then tests the CPUID byte in the PSA to determine whether it is being executed on the CPU to be varied. If so, the routine unlocks the supervisor, sets the timer, and waits.

Once the timer has been set, the VARY CPU task becomes non-dispatchable. No I/O operations are in progress (and since all tasks have been set non-dispatchable, none will be initiated). The timer is in the other CPU, the operator is forbidden to depress the interrupt key, and no shoulder taps should occur. The CPU to be varied is therefore in a wait state, enabled for interruptions; when the timer interruption occurs, it will occur in the other CPU. Thus the other CPU will continue the execution of the VARY CPU task.

When the timer interruption occurs, the supervisor dispatches the VARY CPU task. The VARY CPU routine issues a shoulder tap to the CPU to be varied, which unlocks the supervisor and executes a Diagnose Stop instruction.[1]

When the CPU to be varied unlocks the supervisor, the VARY CPU routine, being executed by the other CPU, locks the supervisor. The VARY CPU routine sets the IEATCBP field in the PSA of the CPU to be varied to point to the dummy high priority task and adjusts the CPUSTAT fields in both PSAs to reflect the new configuration. The routine sets all TCBs dispatchable, writes an SMF record if the SMF function is in the system, issues a WTO macro instruction to inform the operator that the command has been executed, and returns control to the supervisor.

When channel and unit availability has been verified, the Initialization routine sets all TCBs dispatchable and initializes the CPUSTAT byte in both PSAs. It sets the second word of both IEATCBP fields to zero, initializes its external and machine check new PSWs for multiprocessing, and uses the WTO macro instruction to inform the operator that the command has been executed (and to notify him of any channels that are offline). When the WTO macro instruction has been issued, the VARY CPU routine turns off the common bit in the status flags field of the master scheduler resident data area, and returns control to the supervisor.

The VARY Storage Command

The operator issues the VARY Storage command to change the status of an area of storage (except those areas allocated to the supervisor) to online or offline. When

----

[1] The Diagnose Stop instruction, located in the PSA and named VARYSTOP, is followed by an unconditional branch to the Diagnose Stop instruction. Thus, if the Start key is inadvertantly pressed, the system will again stop.

the Command Scheduling routine recognizes the command, it constructs a CSCB and adds the CSCB to the chain. It posts the Master Scheduler, and the Master Scheduler Attach routine uses the ATTACH macro instruction to pass control to the VARY Storage routine.

When it is entered, the VARY Storage routine (module IEEMPVSE) inspects the parameter type switch in the CSCB to determine whether the storage address parameters are expressed as an address range or as a storage box number.

If an address range is specified, the routine converts the numbers to binary form and orders them so that the smaller number becomes the first address and the larger number becomes the second address. The routine then rounds the first address down to the nearest lower multiple of 2048 (2K) bytes, and rounds the second address up to the next higher multiple of 2048 bytes. It then converts the rounded off addresses to EBCDIC form and stores them in the CSCB. Thus the command is made to apply to all 2K blocks of storage encompassed by the specified address range and the operator message written when the command has been executed reflects the actual addresses used.

If a storage box number is specified, the VARY Storage routine computes the second address as 262,144 (256K) times the specified box number. It computes the first address by subtracting 262,144 from the second address.

Offline: When it has determined the addresses of the area to be set offline, the VARY Storage routine ensures that the area does not overlap any main storage areas assigned to the supervisor. It inspects the main storage supervisor partition queue element (MSSPQE), which specifies the lower limit of the main storage available for allocation and the extent of the area. If the first address specified in the command is smaller than the address in the MSSPQE, or if the second address in the command is larger than the address in the MSSPQE plus the length of the area, the area specified in the command includes main storage allocated to the supervisor. In this case, the routine rejects the command.

If the routine accepts the command, it constructs a vary queue element (VQE), which is shown in Figure 40. Then it searches for areas of main storage (within the area specified in the command) that are "free", e.g., not allocated.

Free areas of main storage are represented by free block queue elements (FBQEs), which have the format shown in Figure 41. The routine searches the free block queue for FBQEs whose areas overlap the area specified in the command, and modifies the queue so that such areas are no longer represented by FBQEs.

In order to determine whether the area represented by the FBQE (the free block) overlaps the area specified in the command (the VARY area), the routine compares the beginning and ending addresses of the free block to the beginning and ending addresses of the VARY area, and performs the processing described in Figure 42.

Reserved 1	Pointer to Next VQE 3
Reserved 1	Lower Boundary of Specified Area 3
Reserved 1	Length of Specified Area 3
VARY ECB 4	

Figure 40. Vary Queue Element (VQE)

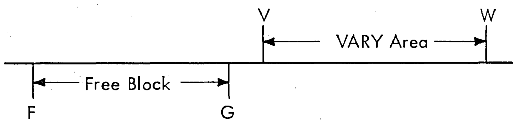
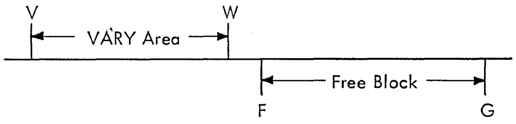
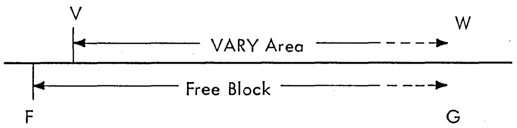
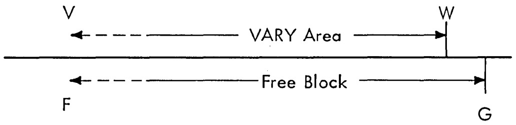
Reserved 1	Forward Pointer
Reserved 1	Backward Pointer
Reserved 1	Number of Bytes in Set of 2K Blocks

Note: The FBQE occupies the first three words of the area it represents.

Figure 41. Free Block Queue Element (FBQE)

When the VARY Storage routine has dequeued an FBQE (or modified one so that an area of storage is no longer represented), it marks the storage area offline in the fail soft storage area map (FSSEMAP). Each PSA contains a copy of the FSSEMAP, which is an area of 2048 bits (64 words). Each 2K block of main storage is represented in the FSSEMAP by two bits: bits 0 and 1 correspond to block 0; bits 2 and 3 correspond to block 1, etc. The two bits are set on if the corresponding block of main storage is offline, and they are set off if the block is online. When it has marked an area of storage offline, the VARY Storage routine continues its search of the free block queue.

When it has terminated the search, the VARY Storage routine examines the FSSEMAP to determine whether the command has been completely executed, or whether there are blocks of storage in the area specified in the command that are allocated as regions and therefore are not represented by an

SITUATION	ACTION
	If this situation exists, the areas do not overlap, but there may be free blocks that do overlap the VARY area; the search continues.
	If this situation exists, the areas do not overlap, and since no succeeding free blocks will overlap the VARY area, the search ends.
	If this situation exists, there is an area at the lower end of the free block that will remain online. The routine builds and enqueues an FBQE that represents the area from V to G, and modifies the original FBQE to represent the area from F to V.
	If this situation exists, there is an area at the upper end of the free block that will remain online. The routine builds and enqueues an FBQE that represents the area from W to G, and deletes the original FBQE (or the one built as a result of situation 4).

Note: The letters V and W represent the beginning and ending addresses of the main storage area specified in the VARY command; the letters F and G represent the beginning and ending addresses of the area represented by the FBQE.

Figure 42. The VARY STOR OFFLINE Command

FBQE. If there are areas of storage that must still be set offline to satisfy the command, the VARY Storage routine issues a WAIT macro instruction specifying the ECB in the VQE. This ECB may be posted by a routine used in the execution of the FREEPART macro instruction (if the region of storage freed overlaps the VARY area), or it may be posted by the VARY Storage routine when it is executing an overriding VARY ONLINE command.

When the ECB has been posted, the VARY Storage routine examines the length field in the VQE. If the length field is zero, the ECB was posted by the VARY Storage routine executing an overriding VARY command. In this case, the VARY Storage routine uses the WTO macro instruction to inform the operator that a VARY command was overridden by a subsequent VARY command, dequeues the VQE, and returns control to the supervisor.

If the VQE length field is not zero, the ECB was posted during execution of the FREEPART routine. In this case, the VARY Storage routine scans the FSSEMAP to determine whether there are any storage blocks specified in the command that were not set offline during execution of the FREEPART routine. If so, the command has still not been executed, and the routine again issues the WAIT macro instruction.

If, however, the entire area specified in the command has been set offline, the command has been executed; the routine uses the WTO macro instruction to inform the operator, dequeues the VQE, reduces the VARY Storage count by one, and returns control to the supervisor.

Online: When it has calculated the addresses bounding the area to be set online, the routine determines whether the command overrides a previous VARY command that has not yet been completely executed. The routine searches the vary queue, looking for a VQE that represents the same address range specified in the current command.

If the routine finds a VQE whose area partially overlaps the area specified in the current command, the VARY Storage routine rejects the current command. If, however, it finds a perfect match, the routine sets the length field in the VQE to zero and posts the ECB in the VQE. This notifies the routine executing the overridden command that the command has been overridden.

When it has issued the POST macro instruction, or if no matching (or overlapping) VQE was found, the routine compares the address range specified in the

VARY command with the address range in the MSSPQE. If the area specified in the command overlaps an area allocated to the supervisor, the routine adjusts the address range to remove the overlap.

When it has made any necessary adjustments, the routine verifies that the storage specified in the command is physically present in the sytem. If the specified storage is not in the system, the routine uses the WTO macro instruction to inform the operator, then returns control to the supervisor.

If the specified storage is in the system, the VARY Storage routine marks each block online in the FSSEMAP, and constructs or extends an FBQE to represent the block.

When each 2K block in the specified area has been brought online, the routine uses the WTO macro instruction to inform the operator that the command has been executed, turns off the common bit in the status flags field of the master scheduler resident data area, and returns control to the supervisor.

EXISTING-TASK COMMANDS

There are certain commands that must be executed by routines performing tasks currently established in the system. The DISPLAY JOBNAMES command, for example, can only be executed by existing initiators. These commands are called existing-task commands.

The CANCEL Command

When an initiator starts to process a job, it creates and enqueues a CSCB to represent the job step. During the process of initiation the CSCB is checked for a cancel indication; if the command is received before the job step's TCB is attached, the job is immediately terminated.

If no CANCEL command is received, the ATTACH macro instruction is issued and the initiator goes into the wait state; it becomes ready again when a program executing the step issues the RETURN macro instruction, or when a CANCEL command causes the Command Scheduling routine to post an ECB in the CSCB for the job step.

Similarly, the DSB Handler routines of the system output writer create a CSCB to represent the data set processing subtask (which may be executed by the Standard Writer routine or by a user-supplied program). When the subtask has been attached, the writer task enters the wait state; it becomes ready again when the program executing the subtask issues a RETURN macro instruction, or when a CANCEL command, using the writer device name as a jobname, causes the Command Scheduling routine to post an ECB in the CSCB that represents the data set processing subtask.

The CANCEL Processor routine (module IGC3703D) of the Command Scheduling SVC 34 routine searches the chain of CSCBs for the one that represents a step of the job to be canceled. If no such CSCB is found, the job has not been selected, and a new task must be established to manipulate the input queue (see "Task-Creating Commands").

If the CSCB is found, it is marked "canceled" and a POST macro instruction is issued to the cancel ECB in the CSCB. If TSO is active, the POST macro instruction is issued with the TJID. The TJID is the non-zero TSO terminal job identification number. If TSO is not in the system, the TJID is zero.

The POST macro instruction causes an SVC interruption. The executing job step will thus lose control of the CPU; since the initiator has a higher priority than the job step, the initiator will regain control first.

When it gets control, the initiator checks the ECBs in the CSCB. If the cancel ECB has been posted, it marks the CSCB to indicate that an abnormal termination (ABTERM) is required, and issues SVC 34, with register 1 negative, and pointing to the CSCB. The Chain Manipulator routine (module IGC0003D) of the Command Scheduling routine branches to the ABTERM routine, which modifies the job step's PSW to request abnormal termination, then returns control to the initiator. The initiator goes into the wait state until control is again passed to it by the terminating job step.

When the initiator goes back into the wait state, the job step regains control. Since, however, its PSW has been altered to request abnormal termination, the ABTERM immediately occurs.

The CANCEL command can be issued from a TSO terminal or from an operator's console to terminate a TSO terminal session or a background job. The processing of a CANCEL command that is issued from the TSO terminal is described in the TSO Command Processor PLMs.

The MONITOR DSNAME Command

This command is stored when the DISPLAY/MONITOR Commands Router routine (module IGC3503D) of the Command Scheduling routine turns on bit 4 in byte MSSSB in the common area of the master scheduler resident data area. (If the MONITOR DSNAME command is issued from a TSO terminal, module IGC3503D turns on bit 1 of byte BAMONITR in the common area of the master scheduler resident data area. TSO processing of the MONITOR DSNAME command is described in the <u>TSO Command Processor PLMs</u>.)

Four routines execute the MONITOR DSNAME command: two allocation routines, AVR and External Action, execute the command to include the data set name in mount messages; the Step Termination and Job Termination routines execute the MONITOR DSNAME command to include the data set name in demount messages. The names of temporary data sets are not displayed. If the MONITOR DSNAME command is in effect, the four routines execute it in the following ways:

- AVR -- The AVR routine locates the first non-temporary data set and requests the Queue Management routine to read the appropriate JFCB into main storage. The data set name (less extraneous blanks) is extracted from the JFCB and placed into the mount message.

- External Action -- Prior to issuing a mount message for a TIOT entry, the External Action routine scans the TIOT for a nontemporary data set that is allocated to the requested volume or device. Such a nontemporary data set is eligible for display if neither deferred mounting nor unit affinity is specified. If an eligible data set is found, the External Action routine requests the queue manager to read the appropriate JFCB into main storage. The External Action routine extracts the data set name from the JFCB and inserts it into the mount message. If no eligible data set can be found, the data set name is omitted from the mount message.

- Step Termination -- The Step Termination routine (module IEFZGST1) requests queue mangement to read the JFCB associated with the TIOT device entry for which a demount message is to be issued. The TIOT is scanned for the first nontemporary data set allocated to the device. If such a data set is found, the volume serial number(s) in the JFCB are compared with the serial number of the volume to be demounted; if a volume serial number is equal to the serial number of the volume to be demounted, termination message module IEFZHMSG extracts the data set name from the JFCB, inserts it into the demount message, and issues a WTO or a TPUT (if the command was issued from a TSO terminal) macro instruction. If no eligible data set can be found, the data set name is omitted from the demount message.

- Job Termination -- The displaying of the data set name in job termination demount messages is restricted to tape. the Job Termination routine executes the MONITOR DSNAME command during the processing of unreceived passed data sets on tape volumes. Prior to constructing a KEEP disposition message for a nontemporary data set, the Job Termination routine extracts the UCB pointer from the PDQ block entry and the volume serial number from the JFCB. If the volume is mounted, termination message module IEFZHMSG extracts the data set name from the JFCB, inserts it into the demount message, and issues a WTO or a TPUT (if the command was issued from a TSO terminal) macro instruction. If no eligible data set can be found, the data set name is omitted from the demount message.

<u>The MONITOR JOBNAMES Command</u>

The DISPLAY/MONITOR Commands Router routine (module IGC3503D) of the Command Scheduling routine stores this command by setting bits in the master scheduler resident data area:

- The job notification bit (BAJN) indicates that the job's name is to be displayed on the requesting console output device when the first step of the job is initiated and when the last step of the job is terminated.

- The time notification bit (set only if the optional keyword T is used in the command) indicates that the time is to be included in the message.

- If the MONITOR JOBNAMES command is issued from a TSO terminal, module IGC3503D sets bit 0 in byte BAMONITR in the common area of the master scheduler resident data area. TSO processing of the MONITOR JOBNAMES command is described in the <u>TSO Command Processor PLMs</u>.

The MONITOR JOBNAMES command is executed in the Allocation Entry routine and in the Job Termination Exit routine.

<u>The DISPLAY R Command</u>

The DISPLAY R command identifies and displays requests for operator action that have not been satisfied: messages requiring a reply, and volume mounting instructions that have not been executed.

The task of executing the DISPLAY R command is performed by the DISPLAY Requests routine (module IEE2903D), which gets control when the DISPLAY/MONITOR Commands Router routine (module IEE3503D) determines that the DISPLAY command has the R operand. The DISPLAY Requests routine operates in the SVC Transient Area.

The DISPLAY R command may be issued from a console or from a TSO terminal. If the command is issued from a console device, the console identifier is passed to the Display Requests routine in the UCMI field of the XSA of the SVRB. If the command is issued from a TSO terminal, the terminal identifier is in the TJID field of the XSA of the SVRB. (See the TSO Command Processor PLMs for a detailed discussion of TSO processing of the DISPLAY R command.)

The DISPLAY Requests routine first looks for operator messages awaiting a reply. It scans the reply queue, extracts the two-byte message ID from each reply queue element (RPQE), and stores the IDs in a 72-byte output buffer.[1]

Next, the DISPLAY Request routine searches for outstanding mount messages. It scans the list of UCBs, and in each UCB it tests the high-order bit in the SRTEDMCT field. If the bit is on, it means that a mount message has been issued, and if the unit is a direct access device, that the correct volume has not yet been mounted. In this case, the routine stores the unit address in the output buffer.

If the unit is a tape device, the identity of the volume is not verified until a data set is opened. The Display Requests routine assumes that if the unit is in the ready state, the mounting instructions have been executed. If the unit is not in the ready state, the operator has not yet mounted a volume, and the routine stores the unit address in the output buffer.

If the high-order bit of the UCB field SRTEDMCT is not on, the Display Requests routine tests the UCBAT1 field. If the field is not zero, an AVR mount message has been issued, but the operator has not yet mounted the requested volume. In this case, the routine sets a switch, and at the end of the display indicates to the operator that an AVR mount message is pending.

When the output buffer is full, the routine uses a WTO macro instruction or, if the DISPLAY R command was issued from a TSO terminal, a TPUT macro instruction to pass the message IDs and unit addresses to the operator. When the scan is complete and the last message has been written, the routine frees the main storage obtained for the buffer and returns control to the supervisor.

The MONITOR SPACE Command

This command is stored when module IGC3503D of the Command Scheduling routine turns on bit 5 in byte MSSSB of the common area of the master scheduler resident data area. (If the MONITOR SPACE command is issued from a TSO terminal, module IGC3503D turns on bit 2 in byte BAMONITR instead of bit 5 in byte MSSSB. TSO processing of the MONITOR SPACE command is described in the TSO Command Processor PLMs.)

The MONITOR SPACE command is executed by the I/O Device Allocation routine and by the Job Termination and Step Termination routines. When a direct access volume is to be dismounted, the External Action routine (module IEFWD000) of the I/O Device Allocation routine, or the Termination Message routine (module IEFZHMSG) tests bit 5 of byte MSSSB. If the bit is on, the routine issues SVC 78, (see the DADSM PLM), to obtain information on the amount of space available on the volume, then includes the information with the demount message.

The DISPLAY SQA Command

An operator uses the DISPLAY SQA (system queue area) command when he wants to know how much free storage remains in the system queue area. When the Display Router routine (in module IGC3503D) recognizes the DISPLAY SQA command, it issues an XCTL macro instruction to give control to module IGC8503D. This module causes disabling of the system while it obtains the low and high boundaries of the system queue area and the amount of free storage available. (This latter value is determined by chaining through the free queue elements (FQEs). See the Main Storage Supervision section of the MVT Supervisor Program Logic Manual.) The module then causes enabling of the system to occur and issues a WTO macro instruction to display the information just obtained.

The MONITOR STATUS Command

This command is stored when the DISPLAY/MONITOR Commands Router routine of the Command Scheduling routine turns on the MSDISPST bit in the master scheduler resident data area. (If the MONITOR STATUS

----

[1] See the MVT Supervisor Program Logic Manual for information on the reply queue and RPQE.

command is issued from a TSO terminal, module IGC3503D turns on bit 3 in byte BAMONITR instead of the MSDISPST bit. TSO processing of the MONITOR STATUS command is described in the TSO Command Processor PLMs.) Each initiator tests the bit when it terminates a job step; if the bit is on, the Step Termination routine uses a WTO or TPUT macro instruction to display the disposition of the data sets used in the step.

The DISPLAY T Command

This command is executed entirely within the Command Scheduling routine. The DISPLAY/MONITOR Commands Router routine (module IGC3503D) issues the TIME macro instruction to obtain the time and date. It then re-formats the information and uses a WTO or TPUT macro instruction to display the time of day and the date to the requesting operator.

The DISPLAY T command may also be issued from a TSO terminal. TSO processing of the DISPLAY T command is described in the TSO Command Processing PLMs.

The MSGRT Command

The MSGRT (MR) command establishes default routine instructions (L=xxx operands) for the DISPLAY, MONITOR, STOPMN and CONTROL commands. When the Router routine (IGC0403D) determines that a MSGRT command is to be executed, it passes control to the MSGRT Command Handler 1 routine (IGC6303D).

The MSGRT Command Handler 1 routine first determines which command and operand has been entered. If the REF operand has been entered, the routine passes control to the MSGRT Command Handler 2 routine (IGC6403D). Otherwise, the MSGRT Command Handler 1 routine scans the command for syntax errors. If errors exist, the routine sets an appropriate message code in the XSA and passes control to the MSGRT Command Handler 2 routine (IGC6403D). This routine determines that a message is to be written and passes control to the MSGRT and CONTROL (MR and K) Message Module 1 routine (IGC5603D), which writes the proper message to the operator's console.

If the MSGRT Command Handler 1 routine does not find syntax errors, it checks the routing location (L=xxx) operands of the command for valid console identification number and display area identifier. If the routine finds invalid operands, it sets an error code in the XSA and passes control to the message module to write a message to the operator's console. If the L=xxx operands are valid, the MSGRT Command Handler 1 routine checks the requesting console's UCM entry to determine if a Message Routing Control Table (MRCT) already exists for that console. If no table exists, the routine issues a GETMAIN macro instruction for the required storage and enters the command operands and routing defaults in the MRCT.

Storage for the MRCT is requested in 48-byte sections. Each table section consists of five 8-byte "data" entries and one 8-byte address entry. Each address entry may point to an additional 48-byte table section (there may be several additional sections; the address entry of the last table section contains zeros). A data entry in the MRCT consists of a 1-byte control byte, a 2-byte command code, a 2-byte operand code, a 1-byte console identifier, a 1-byte display area identifier, and one unused byte. For example, an MRCT entry establishing default routine operands for a DISPLAY Q command (routing all DISPLAY Q commands to console "09" and display area "A") would appear as follows:

```
0    1     3     5      6     7      8
┌────┬─────┬─────┬──────┬─────┬──────┐
│ 01 │  D  │  Q  │  09  │  A  │  00  │
└────┴─────┴─────┴──────┴─────┴──────┘
        8 bytes
```

Each command-and-operand combination for which routine defaults are established comprises one entry in the MRCT.

If the MSGRT Command Handler 1 routine determines that an MRCT already exists for a console, it checks the table to determine if it already contains an entry for the command-and-operand combination that was entered. If a duplicate entry is found, the routine changes the console identification number byte and the display area identifier byte as appropriate to reflect the new default routing instructions. If no duplicate entry is found, the routine must make a new table entry. The routine searches for free space in an existing table and, if space is found, it places the new entry in the existing table. If no space is found, the routine issues a GETMAIN macro instruction for another 48-byte table, and places the entry in this new table section.

MSGRT REF: When the MSGRT Command Handler 1 routine determines that the operator has entered a MSGRT REF command, it passes control to the MSGRT Command Handler 2 routine. This routine locates the MRCT and formats the entries for display. The routine then determines if the entering console is a display (CRT) console. If so, the routine places the table in the entry area of the Display Control Module (DCM) so that it may be written to the screen, and it sets a code in the XSA to cause a display of the "CHANGE OPTIONS" message. If the issuing console was not a CRT, the routine uses the WTO macro instruction to write the display of message routing defaults.

The CONTROL Command

When the Command Verb Checker routine (IGC0403D) determines that a CONTROL command is to be executed, it passes control to the Routing Location module (IGC7503D). This routine scans the command for routing (L=xxx) operands. If complete routing instructions have been specified, it places them in the extended save area (XSA). If no routing operands, or only partial routing operands, have been specified, the routine checks the routing control table for default routing operands. If it finds default values, it places them in the XSA. If no defaults are found, the routine identifies the issuing console as the receiving console and defaults the area identifier to a null value (the first available area will be used). After the routine identifies the console address and display area, it places appropriate indicators in the XSA.

The Command Verb Checker routine then passes control to the CONTROL Command Handler 1 routine (IEE6703D). This routine checks the syntax of the command and, if major operands A, M, V or C have been entered, passes control to the appropriate routine as described in the following sections:

CONTROL A: The CONTROL Command Handler 4 routine (IEE7803D) receives control from the CONTROL Command Handler 1 routine for all CONTROL A commands. This routine checks the syntax of the command. If the syntax is invalid, module IEE7803D passes control to the MSGRT and CONTROL Message Module (IEE5603D). If the CONTROL A command syntax is valid and the command is other than CONTROL A,REF or CONTROL A,NONE, module IEE7803D passes control to the CONTROL Command Handler 2 routine (module IEE6803D). This routine first scans for syntax errors and invalid area definitions.

If any are found, it sets an error code in the XSA and passes control to the MR and K Message Module (IGC5603D), which writes an error message to the operator's console. If the routine finds no errors, it determines whether an existing area is being changed or a new area is being defined. If an operator is changing an existing area, the routine changes the existing screen area control block (SACB) in the resident portion of the display control module (DCM). If an operator is defining a new area, the routine issues a GETMAIN macro instruction so that additional SACBs may be built.

CONTROL A,REF and CONTROL A,NONE: The CONTROL Command Handler 4 Routine (IEE7803D) receives control from the CONTROL Command Handler 1 Routine for all CONTROL A Commands. This module checks the syntax of the command. If the syntax is invalid, the module passes control to the MSGRT and CONTROL Message Module (IEE5603D) for an error message. If the command is a valid CONTROL A,REF or CONTROL A,NONE command, module IEE7803D passes control to the COMMAND Command Handler 5 routine (IEE6903D) to determine which command has been entered.

If a CONTROL A, REF command has been entered, or if a CONTROL A command has been entered without any other operands (this condition defaults to a CONTROL A, REF command condition), the CONTROL Command Handler 5 routine obtains the sizes of all existing display areas from the screen area control block. If the console that issued the command is a display console, module IGC6903D builds a message in the entry area of the DCM and sets a code in the XSA to indicate that the "CHANGE OPTIONS" message also should be written to the operator's console. If the console is not a display console, module IGC6903D uses the WTO macro instruction to write the display.

If a Control A,NONE command (indicating that all display area definitions are cancelled) has been entered, the CONTROL Command Handler 5 routine first determines whether any displays are currently being displayed in the areas that are being cancelled. If so, the routine rejects the command and passes control to the MR and K Message Module (IGC5603D), which writes an appropriate error message to the operator's console. If no displays are in the cancelled areas, the routine reinitializes the screen area control blocks contained within the resident portion of the DCM (these are the blocks defined during system generation). The routine also issues a FREEMAIN macro instruction to release the screen area control blocks that were added (by means of a GETMAIN macro instruction) after system generation.

CONTROL M and CONTROL V: The CONTROL Command Handler 3 routine (IGC7703D) receives control from the CONTROL Command Handler 1 routine whenever a CONTROL M Command or a CONTROL V command is entered.

If the CONTROL V command has been entered, the routine sets flags in the UCM to indicate the new console status, and posts the ECB of the UCM to change the console status.

If the CONTROL M command has been entered, the routine first determines whether the issuing console was the master console (the CONTROL M command can only be entered validly through the master console). If the issuing console was not the master console, the routine passes control to the MR and K Message Module (IGC5603D) for an error message. If the issuing console was the master console, the routine scans the command operands. If operands K M, REF, or K M with no other operands (which defaults to a K M, REF condition) have been entered, the routine either displays the current value for the system timer in the entry area (if the issuing console is a display (CRT) console) or uses a WTO macro instruction to write the current value to the console.

If the command CONTROL M, UTME=nnn has been entered, the CONTROL Command Handler 3 routine determines if the value of the nnn field is valid (it must be a value between 30 and 999). If the specification is valid, the routine changes the time field in the UCM to reflect the new value.

CONTROL C: A CONTROL C command is entered to cancel a status display. The CONTROL Command Handler 4 routine receives control from the CONTROL Command Handler 1 routine whenever a CONTROL C command is entered.

The routine first checks the command syntax and the display identification number for validity. If either is invalid, the routine sets a message code in the XSA and passes control to the MR and K Command Message Module (IGC5603D) for an appropriate error message. If the command is valid, the routine searches the console output queue for a WQE with an identification number that corresponds to the identification number specified in the CONTROL C command. If a corresponding number is found, the routine removes the message from the queue. If one is not found, the routine sets an error code in the XSA and passes control to the MR and K Message Module to write an error message.

Other CONTROL Command Operands: The CONTROL Command Handler 1 routine handles all other CONTROL command operands. The routine schedules commands for execution according to the operands entered, by building a parameter list either in the DCM (for commands routed only to the issuing console) or in an area of main storage for which the routine issues a GETMAIN macro instruction (for commands routed to consoles other than the issuing console). The parameter list consists of the command plus the appropriate display area and console identifiers. Device Independent Display Operator Console Support (DIDOCS) routines (described in IBM System/360 Operating System: MVT Supervisor Program Logic Manual, GY28-6659) perform the actual execution of the commands.

The HALT Command

The HALT command is executed by the Statistical Update routine (see the I/O Supervisor Program Logic Manual) as a result of an SVC 76 issued by the HALT and SWITCH SMF Commands Processor (module IGC1403D) of the Command Scheduling routine.

The MODE Command (Models 85, 145, 155, and 165 Only)

The operator issues the MODE command with Models 85, 145, 155, and 165 to display the status of certain machine functions or to control the mode for recording recoverable machine errors. With the Model 85, the MODE command is also used to reactivate deleted sectors in the high-speed buffer storage area and to reactivate the high-speed multiply circuitry. With the Model 165, the MODE Command is also used to reactivate the high-speed buffer storage area.

When the operator issues the MODE command, the Command Router routine (module IEE0403D) passes control to the MODE Command Router routine (module IGF2603D). The MODE Command Router routine determines the machine model and uses the XCTL macro instruction to pass control to the appropriate Machine Status Control routine. These routines execute the MODE command.

For Model 85 there are two Machine Status Control routines (modules IGF08501 and IGF08502) while for Models 145, 155 and 165 there is only one apiece (modules IGF29701, IGF29601 and IGF55301 respectively). The following paragraphs describe the operation of these five routines:

- Machine Status Control routine for Model 85, Part 1 (module IGF08501) -- This routine receives control from the MODE Command Router routine. If the STATUS parameter is specified in the MODE command, the routine obtains machine status information from the machine status block (MSB) for the Model 85 (see Figure 43) and uses a WTO macro instruction to display the status message on the active system console device. The following data is included in the status message:

  1) HIR mode, threshold, present error count.

2) ECC mode, threshold, present error count.

3) Sector deletions.

4) Status of multiply feature.

5) Status of high-speed buffer.

After processing the STATUS parameter, the routine exits via SVC 3 to the supervisor.

If the STATUS parameter is not specified in the MODE command, IGF08501 passes control to IGF08502 which processes all other command parameters.

- Machine Status Control routine for Model 85, Part 2 (module IGF08502) -- This routine processes five command parameters: INIT, HIR, ECC, SECT, and HSM.

  INIT -- The routine sets the following machine features to their initial status: it enables the high-speed buffer by setting bit 33 in the RESMCW field of the Model 85 MSB to zero; it reactivates previously deleted and repaired sectors of the high-speed buffer by resetting corresponding bits in the RESECT field to zero; it reactivates high-speed multiply circuitry by reinitializing bit 8 in the RESMCW field to zero; it sets the machine recovery facilities, HIR and ECC, to recording mode by setting bits 36-39 of the RESMCW to zero; and it resets the threshold count to the value that was specified at system generation time.

  HIR -- The routine sets the Hardware Instruction Retry facility to either recording mode or count mode. If the operator requests recording mode, the routine sets bits 36 and 37 in the RESMCW field to zero. In recording mode, every machine check error causes an interruption, and the Machine Check Handler routine formats and writes an error message to the operator using a WTO macro instruction. The operator specifies a threshold number for recording mode which is the number of interruptions to be taken before MCH switches to count mode. This number is stored in the RETHIR field of the Model 85 MSB. A count of each recoverable error is kept in the RELHIR field. When a comparison indicates that the threshold has been reached, RMS processes the machine check, and then the switch from recording to count mode is made.

  If the operator requests the count mode, the routine sets bits 36 and 37 in the RESMCW field to zero and one respectively. The routine stores the threshold number for count mode in the RETHIR. A counter in low storage is incremented each time a recoverable machine error occurs, and MCH is not entered until the threshold count is exceeded.

  ECC -- The routine sets the Error Correction Code to either recording mode or count mode. If recording mode is requested, the routine sets bits 38 and 39 in the RESMCW field to zero and places the threshold count in the RETECC field of the Model 85 MSB. The procedure described above for HIR is followed. If count mode is requested, bits 38 and 39 are set to zero and one respectively. The threshold number for count mode is placed in the RETECC and the procedure described above for HIR is followed.

  SECT -- The Machine Status Control routine, Part 2, reactivates previously deleted and repaired sectors of the high-speed buffer by setting corresponding bits in the RESECT field of the Model 85 MSB to zero.

  HSM -- The routine causes bit 8 in the RESMCW field to be reinitialized to zero allowing the high-speed multiply circuitry to be used.

When processing of the command parameter is completed, the routine passes control to the supervisor via SVC 3.

- Machine Status Control routine for Model 155 .(module IGF29601) -- This routine receives control from the MODE Command Router routine when the operator issues the MODE command with the Model 155. The routine determines which of the three command parameters (STATUS, HIR, or ECC) is specified and takes the appropriate action:

  STATUS -- The routine obtains machine status information from the machine status block (MSB) for the Model 155 (see Figure 43), which is pointed to by the CVTRMS field of the CVT. Then the routine uses a WTO macro instruction to display the status message on the system console device. The following data is included in the status message:
  1) HIR mode, current error count, error threshold, elapsed time or 'INVL', time threshold.
  2) ECC mode, current error count, error threshold, elapsed time or 'INVL', time threshold.
  3) Buffer pages deleted.

HIR -- The routine sets the hardware Instruction Retry facility to either recording or quiet mode depending on the parameter specified by the operator. In recording mode, every machine check causes an interruption, and the Machine Check Handler routine issues an error recovery report. The recording mode may be requested with or without specifying threshold values, i.e., the number of errors and the time allowed before the Machine Check Handler routine switches to quiet mode. If threshold values are not furnished by the operator, IBM default values are used. In either case, the routine stores the error threshold and the time threshold in the appropriate MSB fields (see Figure 43). Then the routine places zeros in the error and time counters which are also in the MSB.

In quiet mode, soft machine check interruptions are disabled, and the Machine Check Handler routine does not prepare recovery reports. If the operator requests the quiet mode, the Machine Status Control routine for Model 155 sets bit 4 of control register 14 on. Because of hardware design, if HIR is placed in the quiet mode, ECC is also placed in the quiet mode.

ECC -- The routine sets the Error Correction Code to either recording or quiet mode. As for HIR, the operator may request the recording mode with or without specifying threshold values. The procedure described above for HIR is followed; however, the routine uses different fields of the MSB (see Figure 43). If the operator requests the quiet mode, the routine issues a diagnose instruction to place ECC in the quiet mode and sets bit 0 in the COMTE field of the MSB on.

Machine Status Control routine for the Model 145 (module IGF29701) - - This routine receives control from the MODE Command Router routine when an operator issues a MODE command for the Model 145, and it places the Model 145's main or control storage in either record, quiet, or threshold (for control storage only) mode as specified by he command parameter. The Machine Status Control routine determines which combination of the following parameters has been specified and takes appropriate action:

MAIN -- This parameter specifies main storage.

CNTR -- This parameter specifies control storage.

RECORD, QUIET, and THRES -- These parameters specify machine mode.

Except for the combination MAIN and THRES, the RECORD, QUIET, and THRES parameters can be used in any combination with the MAIN and CNTR parameters. In each case (combination), the Machine Status Control routine uses a DIAGNOSE instruction to place the Model 145 in the specified mode. The routine returns control to the supervisor by using an SVC 3 instruction.

When processing of the command parameter is completed, the routine passes control to the supervisor via SVC 3.

- Machine Status Control routine for Model 165 (module IGF55301) -- This routine receives control from the MODE Command Router routine when the operator issues the MODE command with the Model 165. The routine determines which of the four command parameters (STATUS, RECORD, QUIET, or ENABLE) is specified and takes the appropriate action:

  STATUS -- The routine obtains machine status information from the machine status block (MSB) for the Model 165 (see Figure 43), which is pointed to by the CVTRMS field of the CVT. The routine then uses a WTO macro instruction to display the status message on the system console device. The status message informs the operator whether the Model 165 is operating in recording mode or quiet mode, what the soft error count and soft error threshold are, and whether the buffer is enabled or disabled.

  RECORD -- When this parameter is specified, the routine turns off bit 4 of control register 14 to allow soft machine-check interruptions, sets the record flag in the MSB on, and sets the soft error counter in the MSB to zero.

  QUIET -- When this parameter is specified, the routine sets bit 4 in control register 14 on to suppress soft machine-check interruptions, then turns off the record flag in the MSB.

  ENABLE -- When this parameter is specified, the routine uses the DIAGNOSE instruction to enable the high-speed buffer storage area and resets the buffer error counter in the MSB to zero.

When processing of the command parameter is completed control is returned to the supervisor via SVC 3.

Offset Hex	Dec		
0	0	Maintenance Control Word (RESMCW) 8	
8	8	ECC Recoverable Error Count (RELECC) 4	HIR Recoverable Error Count (RELHIR) 4
10	16	ECC Error Threshold (RETECC) 4	HIR Error Threshold (RETHIR) 4
18	24	Sector Bits (RESECT) 4	28
20	32	Reserved (RESPR1)	
38	56	Used for Emulator Processing 40	

Machine Status Control Block (MSB) for Model 85

Offset Hex	Dec		
0	0	HIR Time Threshold (COMTC) 4	HIR Elapsed Time Counter (COMTBC) 4
8	8	HIR Error Threshold (COMEC) 4	HIR Error Counter (COMSECC) 4
10	16	ECC Time Threshold (COMTE) 4	ECC Time Counter (COMTBE) 4
18	24	ECC Error Threshold (COMEE) 4	ECC Error Counter (COMSECE) 4
20	32	Buffer Pages Deleted (COMBUFPG) 4	Reserved 4
28	40	Address of Master Console (MSBMSCON) 4	Address of Hardcopy Backup Console (MSBHDCPY) 4
30	48	MODE Command Work Area 48	

Machine Status Block (MSB) for Model 155

Offset Hex	Dec		
0	0	Maintenance Control Word (MSBMCW) 8	
8	8	Soft Error Counter (MSBCOUNT) 4	Control Register 14 Work Area 4
10	16	Soft Error Threshold (MSBTHRLD) 4	Buffer Error Count (MSBBUFER) 2 Buffer Error Threshold (MSBBUFTH) 2
18	24	Primary Extended LOGOUT Pointer (MSBPR1) 4	Secondary Extended LOGOUT Pointer (MSBSEC) 4
20	32	Status Field (MSBMODE) 1	Reserved 7
28	40	Master Console UCB Pointer (MSBMSCON) 4	Hard Copy UCB Pointer (MSBHDCPY) 4
30	48	Reserved 48	

Machine Status Control Block (MSB) for Model 165

Figure 43. Machine Status Block (MSB)

The MODIFY Command

The MODIFY command allows the operator to change the list of classes that are processed by direct system output (DSO) processing, a system output writer, or an initiator, and the conditions under which a writer must pause for forms changes. It also allows the console operator to change initiator job classes and to pass command images from remote terminals by RJE. The MODIFY command allows the console operator and TSO user to alter his started TSO job. (See the TSO Command Processor PLMs for a description of TSO processing of the MODIFY command.) The STOP and MODIFY Scheduling routine (module IGC0703D) searches the chain of GCBs, pointed to by the M/S resident data area, for a GCB that corresponds to the specified task. The routine builds a command input buffer (CIB) -- see Figure 44 -- from the command image. If a GCB corresponding to the specified task is found, the routine adds the CIB to the chain pointed to by the CIB pointer in the GCB. Then the routine searches the chain of CSCBs to find the CSCBs that correspond to the specified task. If the counter (CHCIBCTR) in the CSCB indicates that the maximum number of CIBs allowed by the processing task has not been reached, the routine adds the CIB to the chain pointed to by the CIB pointer in the CSCB (CHCIBP). Then, the routine issues the POST macro instruction for the STOP/MODIFY ECB (CHECBP) in the CSCB. This process is repeated for each CSCB with the same task name. If the CIB cannot be added to the chain because the maximum number has been attained, the STOP and MODIFY Scheduling routine scans the CSCBs for one with the same task name. If no such CSCB can be found, or if the CHCIBCTRs in the CSCBs with the same task name have been reached, the routine frees the main storage occupied by the CIB, and the MODIFY command is rejected.

The SET AUTO Command

The AUTO parameter may be used in the SET command only during system initialization. It causes flags to be set in the Master Scheduler IPL routine. These flags are inspected to determine which of the "automatic" commands, which are assembled into the system when it is generated, are to be executed as part of system initialization.

<table>
<tr><td colspan="4">Address of next CIB in Queue 4</td></tr>
<tr><td>Command Verb Code 1</td><td>Length of CIB (in Doublewds) 1</td><td colspan="2">Reserved</td></tr>
<tr><td colspan="2">Cont. 4</td><td colspan="2">ID of TSO Terminal Issuing Command 2</td></tr>
<tr><td>ID of Console Issuing Command 1</td><td>Reserved 1</td><td colspan="2">Length of Data Field 2</td></tr>
<tr><td colspan="4">Data</td></tr>
</table>

Figure 44. Command Input Buffer (CIB)

Note: The STOP and MODIFY Scheduling routine builds the command input buffer. The Command Verb Code field contains the hexadecimal identifier for the command that is issued:

Code	Command
X'04'	START
X'44'	MODIFY
X'40'	STOP
X'0C'	MOUNT

The mapping macro instruction for the command input buffer is IEZCIB.

The SET DATE and SET CLOCK Commands

These commands are stored and the responder notified, when part I of the SET Command routine (module IGC0603D of the Command Scheduling routine) reformats the operands and passes the value to Part II of the Set Command routine (module IGC8603D). Module IGC8603D places the information in general registers, and passes control to the Timer Maintenance routine. The Timer Maintenance routine is described in the MVT Supervisor Program Logic Manual.

The CLOCK and DATE parameters in the SET command are also used by System/370 models to set the time-of-day (TOD) clock. When an internal SET command is issued by the Master Scheduler IPL routine during IPL, or when the operator issues a SET command from the console, Part I of the SET Command routine (module IGC0603D) checks the command syntax and passes control to part I of the SET TOD clock routine (module IGC6503D). This routine calculates both the time at midnight and the difference between the current time and the time being set. It passes these values to part II of the SET TOD clock routine (module IGC6603D) which does the following:

- Sets the TOD clock.

- Updates the date in the CVT (communication vector table).

- Updates TSO and system TQEs (timer queue elements) using values passed from module IGC6503D.

Module IGC6603D then passes control to the timer maintenance routine. (The Timer Maintenance routine is described in the MVT Supervisor Program Logic Manual.)

If TSO is included in the system, the SET TOD Clock routine passes control via SVC 95 to the TSO module IKJEATI0 so that TSO TQEs can be updated.

The SET PROC and SET Q Commands

The operator may enter SET commands with the operands PROC and Q only when initializing the system after IPL. These operands cause the SVC 34 SET Commands routine to store the locations of the procedure library and the queue data set, respectively, in the master scheduler resident data area. If the keyword "F" accompanies the Q parameter, the routine sets a switch that causes the Queue Management Initialization routine to format the queue data set.

The STOPMN DSNAME Command

When the Router routine (module IGC0403D) determines that a STOP command has been issued, it passes control to the Periodic Stop Command Processor routine (module IGC4503D), which determines whether the command was issued from a console device or from a TSO terminal. If the command was issued from a terminal, module IGC4503D passes control to the MCS/TSO Periodic Stop Command Processor routine (module IGC5503D). (See the TSO Command Processor PLMs for a description of TSO processing of the STOP DSNAME command.)

If the command was issued from a console device, module IGC4503D stores the command by turning off bit 4 in byte MSSSB in the common area of the master scheduler resident data area. This bit is tested by AVR and External Action routines prior to issuing mount messages; it is tested by the Step Termination routine and the Job Termination routine prior to issuing demount messages. If the bit is on, eligible nontemporary data sets are displayed in the messages. For additional discussion of the processing performed, see the topic "The MONITOR DSNAME Command."

The STOPMN JOBNAMES Command

When the Periodic Stop Command Processor routine (module IGC4503D) receives control to store a STOP JOBNAMES command, the routine determines whether the command was issued from a console device or from a TSO terminal. If the command was issued from a TSO terminal, module IGC4503D passes control to the MCS/TSO Periodic Stop Command Processor routine (module IGC5503D). (See the TSO Command Processor PLMs for a description of TSO processing of the STOP JOBNAMES command.)

Module IGC4503D then determines if MCS was included in the system. If not, the module stores the command by turning off the BAJN bit in the master scheduler resident data area. If MCS is included in the system, the module passes control to module IGC5503D, which turns off both the UCMMSGA bit in the UCME for the console receiving the display and the appropriate bit in the master scheduler resident data area.

Initiators test tne master scheduler resident data area bit when they select a job for processing, and when they terminate the last step of a job. If the bit is on, the name of the job is displayed on the requesting operator's console. For additional discussion of the processing performed, see the topic "The MONITOR JOBNAMES Command."

The STOPMN SPACE Command

When the Periodic Stop Command Processor routine (module IGC4503D) receives control to store a STOP SPACE command, the routine determines whether the command was issued from a console device or from a TSO terminal. If the command was issued from a TSO terminal, module IGC4503D passes control to the MCS/TSO Periodic Stop Command Processor routine (module IGC5503D). (See the TSO Command Processor PLMs for a description of TSO processing of the DISPLAY SPACE command.)

If the command was issued from a console device, module IGC4503D stores the command by turning off bit 5 of byte MSSSB in the common area of the master scheduler resident data area. This bit is tested by the I/O Device Allocation routine and the Termination routine when a direct access volume is to be dismounted; if the bit is on, information on the amount of space available on the volume is displayed with the dismount message. For additional discussion of the processing performed, see the topic "The MONITOR SPACE Command."

The STOPMN STATUS Command

When the Periodic Stop Command Processor routine (module IGC4503D) receives control to store a STOP STATUS command, the routine determines whether the command was issued from a console device or from a TSO terminal. If the command was issued from a TSO terminal, module IGC4503D passes control to the MCS/TSO Periodic Stop Command Processor routine (module IGC5503D). (See the TSO Command Processor PLMs for a description of TSO processing of the STOP STATUS command.

Module IGC4503D then determines if MCS was included in the system. If not, the module stores the command by turning off the MSDISPT bit in the master scheduler resident data area. If MCS is included in the system, the module passes control to module IGC5503D, which turns off both the UCMMSGB bit in the UCME for the console receiving the display and the appropriate bit in the master scheduler resident data area.

Each initiator tests the master scheduler resident data area bit when it terminates a job step; if the bit is on, the Step Termination routine uses the WTO macro instruction to display the disposition of the data sets used in the step. For additional discussion of the processing performed, see the topic "The MONITOR STATUS Command."

The STOPMN SESS Command

When the Periodic Stop Command Processor receives control to store a STOPMN SESS command, the routine determines whether the command was issued from a console device or from a TSO terminal. If the command was issued from a TSO terminal, module IGC4503D passes control to the MCS/TSO Periodic STOP/STOPMN Command Processor (module IGC5503D). (See the TSO Command Processor PLMs for a description of TSO processing of the STOPMN SESS command.)

If the command was not issued from a TSO terminal, module IGC4503D then checks to see if MCS is included in the system. If it is not, module IGC4503D stores the command by turning off the BAMSESSC bit in the master scheduler resident data area. If MCS is included in the system, module IGC4503D passes control to module IGC5503D, which turns off the UCMMSGF bit in the receiving console's UCME and the BAMSESSC bit in the master scheduler resident data area.

The master scheduler bit is tested by TSO routines whenever a terminal user logs on or off. If the bit is on, the user-identifier of the terminal user is displayed on the operator's console (for more information about this function, refer to the description of MONITOR SESS in the TSO Command Language SRLs).

The STOP Command

The STOP and MODIFY Scheduling routine (module IGC0703D) searches the chain of GCBs, pointed to by the M/S resident data area, for a GCB that corresponds to the specified task. The routine builds a command input buffer (CIB) -- see Figure 44 -- from the command image. If a GCB corresponding to the specified task is found, the routine adds the CIB to the chain pointed to by the CIB pointer in the GCB. Then the routine searches the chain of CSCBs for the CSCB corresponding to the specified task. It overlays the UCM entry indicator (UCMI) of the console that issued the START command with the UCMI of the console that issued the STOP command, and issues a POST macro instruction specifying the STOP/MODIFY ECB in the CSCB.

It also sets the CHCIBCTR field of the CSCB to zero, indicating that no more MODIFY commands can be accepted.

During its operating cycle, each system task tests the STOP ECB. If it has been posted, the routine returns control to the System Task Control routine.

The STOPMN A Command

When the STOPMN/STOP Command Handler Routine (IGC4503D) determines that a STOPMN A command is to be executed, it passes control by means of a XCLT macro instruction to the Routing Location routine (IGC7503D).

The Routing Location routine scans the command for the routing (L=xxx) operands. If explicit routing operands have been specified, the routine places them in the extended save area (XSA). If no operands, or only partial operands, are specified, the routine checks the Message Routing Control Table (MRCT) for default routing operands. If the routine finds no defaults, it identifies the issuing console as the console requiring termination of a MONITOR A display and defaults the area identifier to a null value. After the routine determines the console address and display area identifiers, it places them in the XSA. It then passes control to the Routing Location routine, load 2, (IGC7603D). This routine determines that a valid STOPMN command has been entered. Routing Location routine, load 2, then sets indicators in the screen area control block

(SACB) of the issuing console's DCM and in the unit control module entry (UCME). The routine then issues an XCTL macro to pass control to the STOPMN Command Handler, load 2, (IGC5503D).

Module IGC5503D terminates the action initiated by the MONITOR A command by turning off the UCMMSGC bit in the unit control module entry for the console on which the display is to be terminated. The routine then determines if other MONITOR A displays are in progress (by checking for a UCMMSGC indicator in other UCME's). If no indicator is found, module IGC5503D turns off the busy bit in the master scheduler resident data area and passes control to the CONTROL Command Handler 1 routine (IGC6703D). Module IGC6703D builds a parameter list and sets flags in the DCM of the console from which the display is to be removed. The parameter list and flags indicate to the supervisor routines that the display is to be erased from the screen (the routines that perform this function are the DIDOCS routines, documented in the MVT Supervisor PLM, GY28-6659). Module IGC6703D then returns control to the supervisor.

The SWITCH Command

The operator uses the SWITCH command to transfer SMF recording. If direct access devices are being used, SMF recording is transferred from the primary to the alternate SMF data set. If SMF recording is on magnetic tape, the SWITCH command causes the current tape volume to be dismounted, and the system then requests and waits for another tape volume to be mounted so as to continue recording. When the Command Router routine recognizes the SWITCH command, it passes control to the HALT and SWITCH Commands Processor routine (module IGC1403D). This routine issues SVC 83 to pass control to the SMF SVC routine. The SMF SVC routine purges the SMF buffers, writes the data records into the SMF data set, then transfers SMF recording using the device status and device address fields in the SMCA. A detailed description of the above processing is in the section "The SMF SVC Routine (SVC 83)" in part 6 of this publication. After the SMF SVC routine completes the processing of the SWITCH command, it returns control to the HALT and SWITCH Commands Processer routine, which issues an XCTL macro instruction to give control to the message module (IGC0503D). A "no message" indication is also given to the message module. Since module IGC0503D is not to issue a message, it directly returns control to the communications task by means of a branch instruction using the contents of register 14.

The UNLOAD Command

When the Router routine (module IGC0403D) of the Command Scheduling routine determines that an UNLOAD command has been issued, control is passed to the VARY/UNLOAD Syntax Scan routine (module IGC1103D). Module IGC1103D passes control to the VARY/UNLOAD Processor (module IGC3103D).

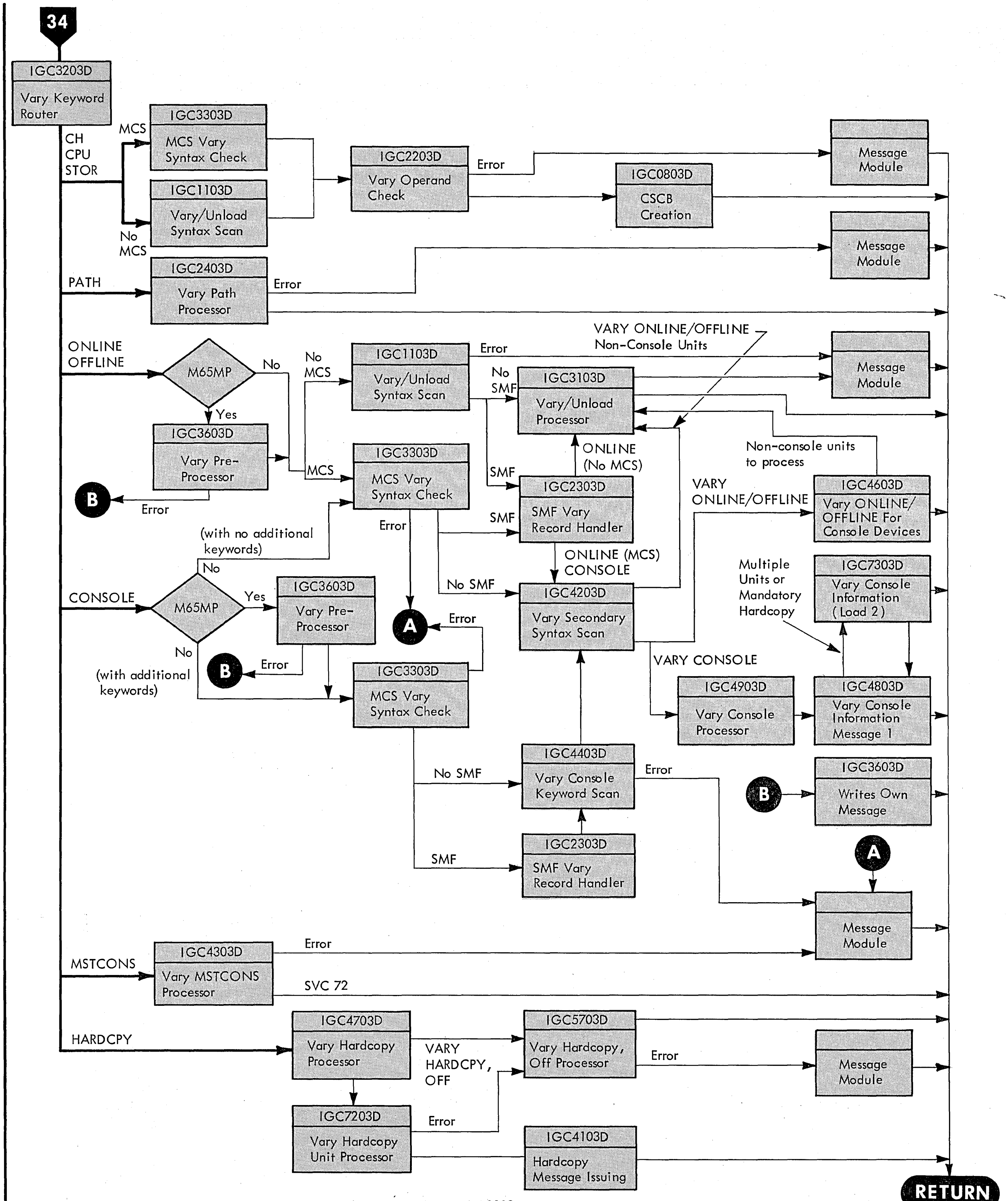
The UNLOAD command is stored when module IGC3103D sets on the SRTEUNLD bit in the UCB of the device to be unloaded. If the unit is not allocated, the routine turns on the BAVU bit (UCB search bit) in the master scheduler resident data area, then returns control to the supervisor. The I/O Device Allocation routine and the Termination routine (performing any system task) test the bits at appropriate points in their processing cycles. If the bits are on, the routine physically unloads the unit.

The VARY Commands

If the command to be executed is a VARY command, the Router routine of the Command Scheduling routine passes control to the VARY Keyword Router routine (module IGC3203D). This routine identifies the first keyword and passes control to the appropriate keyword processing routine (see Figure 45). The keyword processing routines examine the command for syntax errors, and if an error is found, they pass control to a message module that informs the operator and returns control to the supervisor. If the routines find no syntax errors, they process the command.

VARY ONLINE/OFFLINE (Systems Without MCS)

If a system includes the M65MP option, the Vary Keyword Router routine, in module IGC3203D, passes control to module IGC3603D, the Vary Preprocessor routine. This latter module prevents devices that have no paths marked as available to the I/O Supervisor from becoming 'online.' Depending on the absence or the presence of the M65MP option, either module IGC3203D or module IGC3603D then passes control to the VARY/UNLOAD Syntax Scan routine (module IGC1103D). This module scans the VARY ONLINE and VARY OFFLINE commands for syntax errors and passes control to the VARY/UNLOAD processor routine (module IGC3103D). The processor routine searches the list of UCBs for those corresponding to the units specified in the command, and sets the bits that request the specified status change.



Note: The Message Module blocks represent either IGC0503D or IGG2103D.

Figure 45. Control Flow in the Command Scheduling Routine (Part II -- See Figure 24 for Part I)

If any of the units has a syntax error, control passes to a message routine that informs the operator and returns control to the supervisor. If there is no syntax error, the VARY/UNLOAD Processor routine determines if the Online Test Executive Program (OLTEP) is currently running an online test program against a unit specified in the command. If so, the test program must be completed before the unit can be brought online. The processor routine issues an informational message to the operator and returns control to the supervisor. If OLTEP is not running a test program, the VARY/UNLOAD processor routine returns control directly to the supervisor.

The I/O Device Allocation routine and the Termination routine scan the UCBs as a part of their normal processing. When they encounter a UCB for which a status change from ONLINE ALLOCATED to OFFLINE has been requested, they set the bits that indicate the new status.

VARY ONLINE/OFFLINE (Systems With MCS)

If a system includes the M65MP option, the Vary Keyword Router routine, in module IGC3203D, passes control to module IGC3603D, the Vary Pre-processor routine. This latter module prevents devices that have no paths marked as available to the I/O Supervisor from becoming 'online'. When the MCS VARY Syntax Check routine (module IGC3303D) is entered from either module IGC3603D (for M65MP) or module IGC3203D (without M65MP), it scans the unit field for syntax errors. If the routine finds an error, it passes control to a message routine that informs the operator and returns control to the supervisor.

If no errors are found, the routine determines whether SMF is in the system. If so, it passes control to the SMF VARY Record Handler routine (module IGC2303D). Then, or if SMF is not included, control is passed to the VARY Secondary Syntax Scan routine (module IGC4203D). This routine determines whether the command source was authorized to issue the command. It also determines whether the units are valid system units. If errors are found, the routine issues error messages.

The routine also determines whether the specified unit is either the hard copy log device or the master console. If so, it issues a message to the operator informing him that the unit cannot be processed.

If the unit can be processed, the VARY Secondary Syntax Scan routine passes control to the VARY ONLINE/OFFLINE Processor for Console Devices routine (module IGC4603D). This routine determines whether a DISPLAY JOBNAMES command is in force for the unit. If so, and if no DISPLAY JOBNAMES is in force for any other unit, the VARY ONLINE/OFFLINE Processor for Console Devices routine turns off the BAJN bit in the master scheduler resident data area.

The processor routine then determines if OLTEP is currently running an online test program against the unit that is to be processed. If so, the test program must be completed before the unit can be brought online. The processor routine issues an informational message to the operator and returns control to the supervisor.

If OLTEP is not running a test program, the VARY ONLINE/OFFLINE Processor for Console Devices routine sets the bits in the UCM entry and UCB corresponding to the unit so that they indicate the new status. It determines whether the VARY Secondary Syntax Scan routine has encountered any non-console units, and whether the entire unit field has been scanned:

- If there are additional unit specifications to be scanned, the routine processes them.
- If the entire unit field has been scanned and non-console units have been encountered, the routine passes control to the VARY/UNLOAD Processor routine (module IGC3103D).
- If the entire unit field has been scanned and no non-console units have been encountered, the routine returns control to the supervisor.

If the VARY Secondary Syntax Scan routine scans the entire unit field without encountering a console unit it passes control to the VARY/UNLOAD Processor routine (module IGC3103D). This routine performs the processing described under "VARY ONLINE/OFFLINE (Systems Without MCS)" and returns control to the supervisor or (if it encounters an error) passes control to a message module.

VARY PATH

With Alternate Path Retry (APR), the operator may issue the VARY PATH command to cause a path to be logically brought online for use by the system or logically removed from the system.

The VARY PATH Processor routine (module IGC2403D) is entered from the VARY Keyword Router routine. The routine causes the path to be brought online or taken offline by setting the path status bits in the UCB using the IOSGEN macro instruction.

The last path to a device will not be varied offline because this may be accomplished by varying the device itself. (A reserved path to a shared direct access storage device is considered to be the last path.) Also, teleprocessing paths cannot be varied.

Upon completion, the VARY PATH Processor routine passes control to one of two message modules (module IGC0503D or module IGG2103D).

The VARY PATH command is also discussed in the I/O Supervisor PLM.

VARY CONSOLE

If a system includes the M65MP option, the VARY Keyword Router routine, in module IGC3203D, passes control to module IGC3603D, the Vary Preprocessor routine. This latter module prevents devices that have no paths marked as available to the I/O Supervisor from becoming an active console. When the MCS VARY Syntax Check routine (module IGC3303D) is entered from either module IGC3603D (for M65MP) or module IGC3203D (without M65MP) to process a VARY CONSOLE command, it determines whether there are additional keywords in the command. If so, it passes control to the VARY CONSOLE Keyword Scan routine (module IGC4403D); if not, it passes control directly to the VARY Secondary Syntax Scan routine (module IGC4203D). If SMF is included in the system, the MCS VARY Syntax Check routine first passes control to the SMF VARY Record Handler routine (module IGC2303D), which will then pass control to either the VARY CONSOLE Keyword Scan routine or the VARY Secondary Syntax Scan routine.

The VARY CONSOLE Keyword Scan routine determines whether the additional keywords are specified correctly. If not, it passes control to a message routine that informs the requesting operator and returns control to the supervisor. If the additional keywords are correctly specified, or if there were no additional keywords, control passes to the VARY Secondary Syntax Scan routine (module IGC4203D).

The VARY Secondary Syntax Scan routine determines whether the issuing console is authorized to issue the command. If not, the routine informs the requesting operator and returns control to the supervisor.

If the command is authorized, the VARY Secondary Syntax Scan routine checks the first unit specification for syntax errors. If it finds an error, it issues a message to the operator. If no errors are found, it passes control to the VARY CONSOLE Processor routine (module IGC4903D).

The VARY CONSOLE Processor routine maintains the console configuration. The routine then determines if the OLTEP is currently running an online test program against the specified unit. If so, the unit cannot be added to the active console configuration until the test program completes its processing. Module IGC4903D issues a header message and passes control to the Console Information Message routine (module IGC4803D), which issues a status message. The VARY CONSOLE Processor routine then returns control to the supervisor.

If OLTEP is not running a test program, the VARY CONSOLE Processor routine uses information provided through the VARY command to change the attributes of active consoles (the console's routing code assignment, command authority, and alternate console), and brings consoles being used for other purposes (online or offline) into active console status. In addition, it insures that a console log is active and receiving the minimum required routing codes whenever the log is mandatory.

The VARY CONSOLE Processor routine then issues a header message and passes control to the Console Information Message load 1 routine (module IGC4803D) to construct a message that states the unit's new attributes. Module IGC4803D passes control to the Console Information routine load 2 (module IGC7303D). This routine completes the message, uses the WTO macro instruction to issue it to the operator, and returns control either to IGC4803D (if multiple units were specified or if the hardcopy log is required) or to the supervisor if processing is complete.

VARY HARDCPY

If the VARY Keyword Router routine encounters a VARY HARDCPY command, it passes control to the VARY HARDCPY Processor routine (module IGC4703D). Module IGC4703D first determines whether the command was issued with proper authorization, i.e., by an operator via the master console or by the system. If the command was not issued with proper authorization, the module passes control to the VARY HARDCPY, OFF Processor routine (module IGC5703D). This module determines the type of error involved and passes control to a message module that informs the operator of the error and returns control to the system.

If the VARY HARDCPY command was entered through the master console or by the system, module IGC4703D determines which operands have been entered. If VARY

HARDCPY, OFF has been entered, the module passes control to the VARY HARDCPY, OFF Processor (IGC5703D).

The processor determines whether the hardcopy log is required; if it is required, the VARY HARDCPY command is invalid, and IGC5703D passes control to a message module for an error message. If the hardcopy log exists but is not required (the hardcopy log is required if there is more than one console in a system or if the only console in the system is a display console), module IGC5703D varies the hardcopy log out of the system and passes control to the Message Assembly routine (IGC2103D), which notifies the operator of the change in status of the hardcopy log.

If a command other than VARY HARDCPY, OFF has been entered, module IGC4703D checks to see if any of the operands CMDS, INCMDS, STCMDS, NOCMDS, or ROUT= has been specified and then sets a flag to indicate which operand has been entered. Module IGC4703D passes control to the VARY HARDCPY UNIT Processor (module IGC7203D), which insures that the specified unit is eligible as a hardcopy log device. If an error is found, the routine passes control to IGC5703D, which passes control to the message module that informs the operator of the error. If a graphics device is in use as a console (a display console), or if there is more than one console in the system, module IGC7203D adds routing codes 1 through 4, 7, 8, and 10 to those already specified in the UCM. The module then sets the commands flag to CMDS if the hard copy log was not previously specified by either CMDS, STCMDS, or INCMDS (NOCMDS is not valid when the hardcopy log is mandatory).

After module IGC7203D sets the attributes of the hardcopy log, it passes control to the Hardcopy Message routine (module IGC4103D). This routine issues to the operator a message that shows the attributes of the hardcopy log and then returns control to the supervisor.

VARY MSTCONS

When the VARY Keyword Router routine encounteres a VARY MSTCONS command, it passes control to the VARY MSTCONS Processor routine (module IGC4303D). This routine performs a command authorization check and a syntax check. The VARY MSTCONS command may be issued only by the system or by the master console, except that if the master console has failed and its chain of alternates is exhausted, the command may be issued via a secondary console.

If the command is authorized and the VARY MSTCONS routine finds no syntax errors, it issues SVC 72 to have the console switch performed, and then returns control to the supervisor. Otherwise, the routine passes control to a message routine that informs the operator and returns control to the supervisor.

Part 6: Common Elements of Job Management

In the performance of job management tasks, Job Management routines frequently refer to or execute system elements that are also used by other Job Management routines. The major common elements of job management are described below:

- The work queues are the temporary storage areas that permit work to be stored in input sequence, and to be processed in priority sequence. They act as buffers between the interpreters, initiators, and system output writers, allowing each to process at maximum speed.

- The Interpreter routine is used to execute reading, automatic SYSIN batching (ASB), command execution, and remote job entry (RJE) tasks. It reads input stream and procedure library statements, converts them to tabular format, and places the tables in the work queue data set.

- The I/O Device Allocation routine is used primarily by the initiator in satisfying the I/O requests of job steps; it is also used by the command processing routines to allocate I/O devices to system tasks.

- The Termination routine is used whenever a job step or system task terminates. It directs the disposition of the data sets used in the execution of the task, and releases the I/O devices for allocation to other tasks.

- System Management Facilities is an optional feature of the control program that provides the means for gathering and recording the type of information that can be used to evaluate system usage. The routines discussed in this section are those that execute the SMF task and the SMF SVC routine, which writes records in the SMF data set.

- The Write-to-programmer (WTP) Facility is a feature that enables the operating system to provide the user with information about his job in the event of an abnormal occurence during execution; the job information is conveyed via messages to the system message class output data set.

The Work Queues

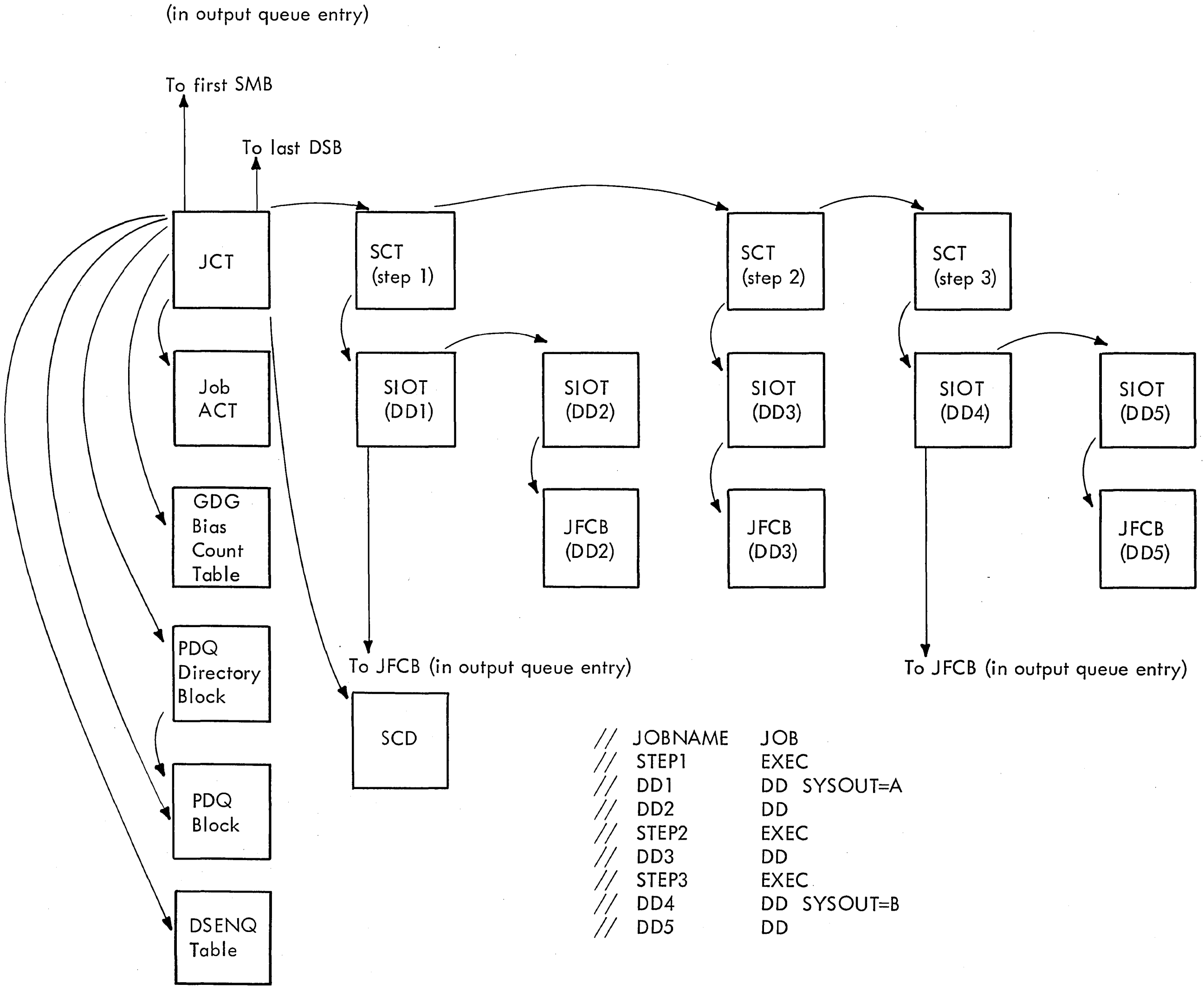
Reading job descriptions, initiating job steps, and writing system output data are the input, processing, and output functions of job management. The work queues allow job descriptions that are read in "random" sequence to be processed according to user-specified priority, and act as buffers to compensate for the differences in speed among the reading, initiating, and writing functions.

At the MVT level of the operating system, there are 76 work queues (the hold queue, the ASB queue, the 36 output queues, the RJE queue, the 15 input queues, the free-track queue, the background reader (BRDR) queue, and 20 reserved queues). As a job description (in JCL form) is processed by the interpreter, it is converted into a series of tables which become the job's input and output queue entries. The input queue entry is enqueued when the interpreter processing of the job is complete. It is enqueued in the queue and at the priority specified in the JOB statement. The format of a typical input queue entry is shown in Figure 46.

Within an input class, jobs are normally selected for processing in priority order (entries of the same priority are dequeued in FIFO order). However, the operator may prevent a job from being selected by issuing a HOLD command; the job description is dequeued from the input queue and enqueued in the hold queue. A RELEASE command dequeues a job description from the hold queue and re-enqueues it in the input queue, where it is again available for selection (in priority order) by an initiator.

System output classes are used to segregate different kinds of system output. For example, messages may be in one class (the message class), data sets to be punched in another, and data sets to be printed in a third. At the MVT level of the operating system, 36 system output classes are provided; there is an output queue that corresponds to each class. The output queues permit system output work to be stacked, then retrieved in the order governed by the job's priority; they act as buffers between the system output writers and the other tasks in the system.

An entry in an output queue represents the system output (of the corresponding class) to be written for a job. If the class is the message class, the entry will contain messages from the operating system to the programmer (in system message blocks, or SMBs), and may also contain data set blocks (DSBs) and job file control blocks (JFCBs), which point to data sets specified as system output data sets of that class.

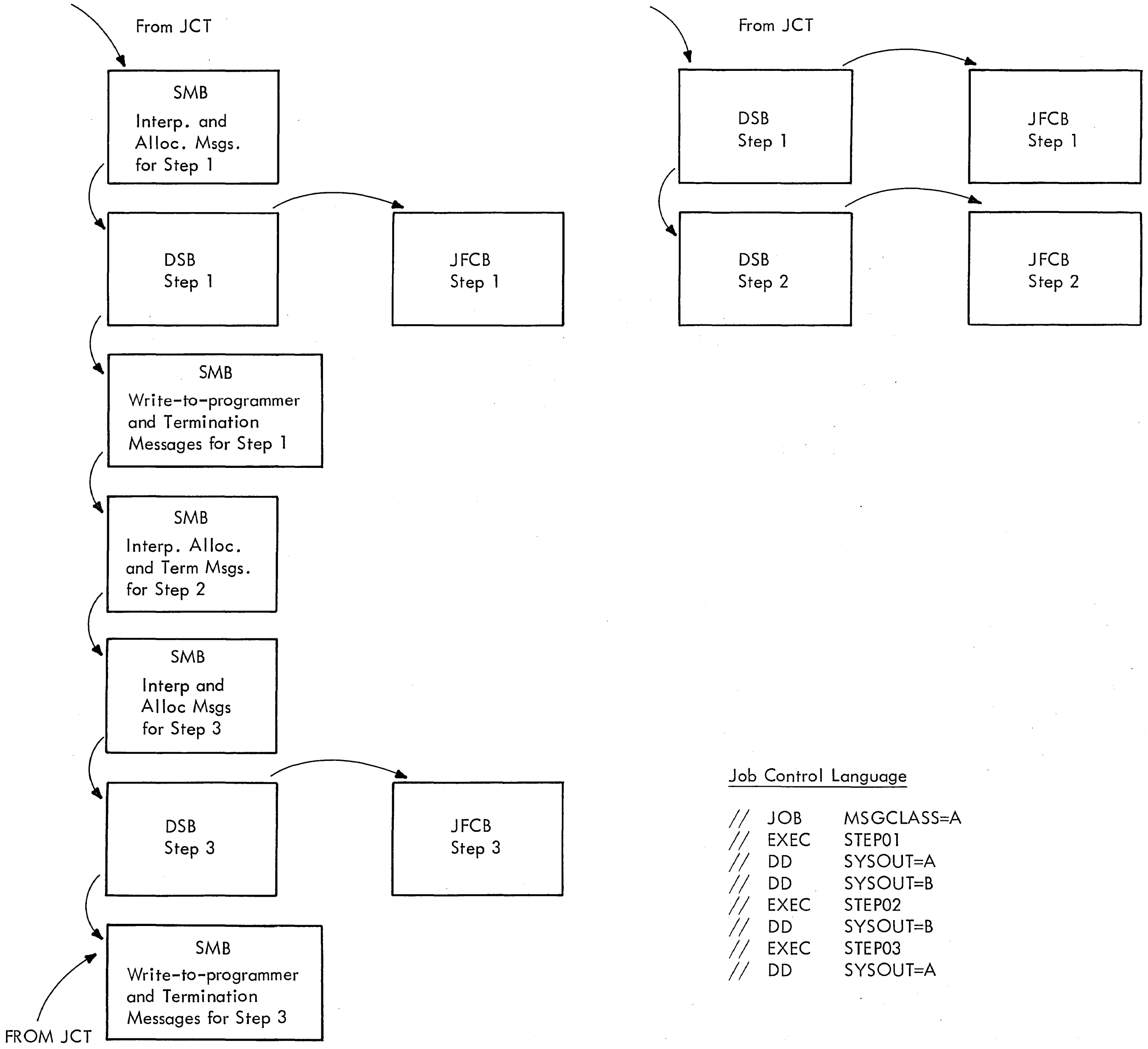


```
// JOBNAME   JOB
// STEP1     EXEC
// DD1       DD SYSOUT=A
// DD2       DD
// STEP2     EXEC
// DD3       DD
// STEP3     EXEC
// DD4       DD SYSOUT=B
// DD5       DD
```

Figure 46. Typical Input Queue Entry

Figure 47 shows two output queue entries, one of which belongs to the message class. System output class A has been designated the message class for the job; the queue entry contains SMBs, and since there are data sets that have been specified to be in class A, the entry also contains DSBs and JFCBs that point to the data sets. (The SIOTs for these data sets are in the job's input queue entry.) If there were no such data sets, the entry would consist entirely of SMBs. The other entry shown in the figure is not in the message class queue, and therefore contains no SMBs.

The list of the logical tracks not assigned to one of the other queues is called the free-track queue. As space for records is assigned to the other queues, additional logical tracks are assigned from the free-track queue. As entries are deleted from the other queues, logical tracks become free; they are assigned to the free-track queue.



Job Control Language

```
// JOB    MSGCLASS=A
// EXEC   STEP01
// DD     SYSOUT=A
// DD     SYSOUT=B
// EXEC   STEP02
// DD     SYSOUT=B
// EXEC   STEP03
// DD     SYSOUT=A
```

Figure 47. Output Queue Entries

QUEUE MANAGEMENT

There are two sets of queue management routines. One set is referred to simply as the Queue Management routines and is loaded with job management modules. These routines assign space in the queue data set, read and write records, enqueue and dequeue work queue entries, and delete them (release the space assigned to them).

The other set of queue management routines is called the Transient Queue Management routines (SVC 90). This set of routines, which is obtained by issuing SVC 90, resides in SYS1.SVCLIB and operates in the transient area. These routines assign logical tracks, read and write job queue records, and assign records to a queue entry, but they do not include the track stacking facility.

When the interpreter encounters a new job in the input stream, it uses queue management to obtain space in the queue data set for the records which make up the job's queue entries. The Queue Management Assign/Start routine performs initializing functions, and the Queue Management Assign routine assigns a logical track; that is, an area of contiguous space in the queue data set containing a header record and space for a predetermined number of 176-byte data records. From that logical track, it assigns an initial number of records.

As the interpreter processes the job, it converts the job control language statements in the input stream into 176-byte records, and uses the Queue Management Read/Write routine to write them into the record areas assigned from the logical track. When the initial quantity of record areas assigned has been filled, the interpreter uses the Queue Management Assign routine to assign additional record areas from the logical track.

When the first logical track assigned has been filled, the Queue Management Assign routine assigns another logical track to satisfy the interpreter's request for additional record areas, and begins assigning space from the new logical track. Figure 48 shows two logical tracks, which are assigned to an entry. The first logical track is fully assigned; its header record points to the second, from which space for four records has been assigned, and into which records have been written.

The pointers among the logical track header (LTH) records are created and maintained by Queue Management routines. The interpreter, the initiator, and the system output writer create and maintain pointers among the 176-byte records so that a record may be retrieved without knowing its position relative to an LTH. The arrangement of pointers shown in Figures 48 through 55 is imaginary; Figures 46 and 47 show the manner in which records are actually chained.

In addition to generating records that describe the job, the interpreter (and later the initiator) must generate records containing messages to the programmer and assign space for records that describe the job's system output data sets. Each time it is necessary to create a record for a new system output class, the interpreter uses the queue management assign/start and assign routines to start a new set of logical tracks by obtaining a new logical track and assigning records from that track.

Meanwhile, as job description records are generated, they are written into space assigned from the original set of logical tracks.

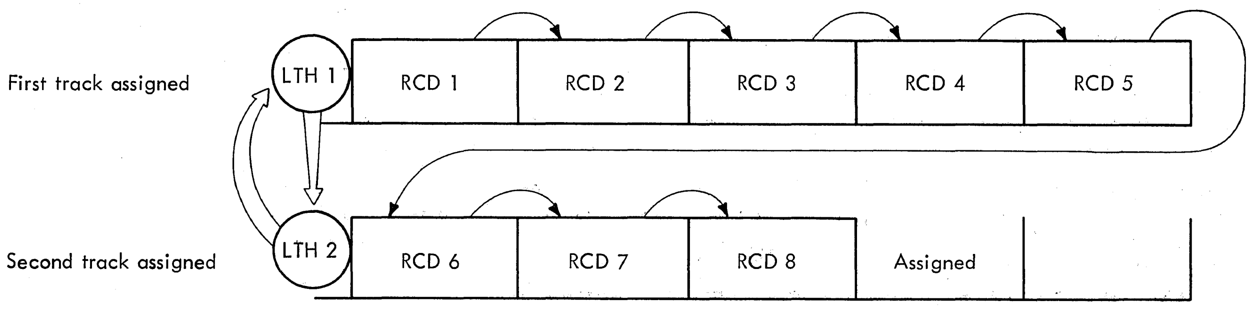


Figure 48. Assigning Space Within Logical Tracks

In Figure 49, an additional logical track has been assigned to the original set, and two records have been written into it. As a result of executing the assign/start and assign routines, two additional logical tracks have been assigned to the entry.

Notice that there are no LTH pointers between the first set of logical tracks and the second. However, in actual practice there are pointers (created by the interpreter) between the 176-byte records assigned to the first set of tracks and the 176-byte records assigned to the second set. The second set contains records that represent one class of system output for the job. Several records (containing messages to the programmer) have been written; space has been reserved for another record, which will be created when the job terminates, and will describe a system output data set.

When the interpreter has completed the processing of a job (all of the records generated by the interpreter have been written into the entry) it uses the Queue Management Enqueue routine to place the entry in an input queue. When an entry is enqueued, it is added to the chain of entries pointed to by the queue control record (QCR) for that queue.

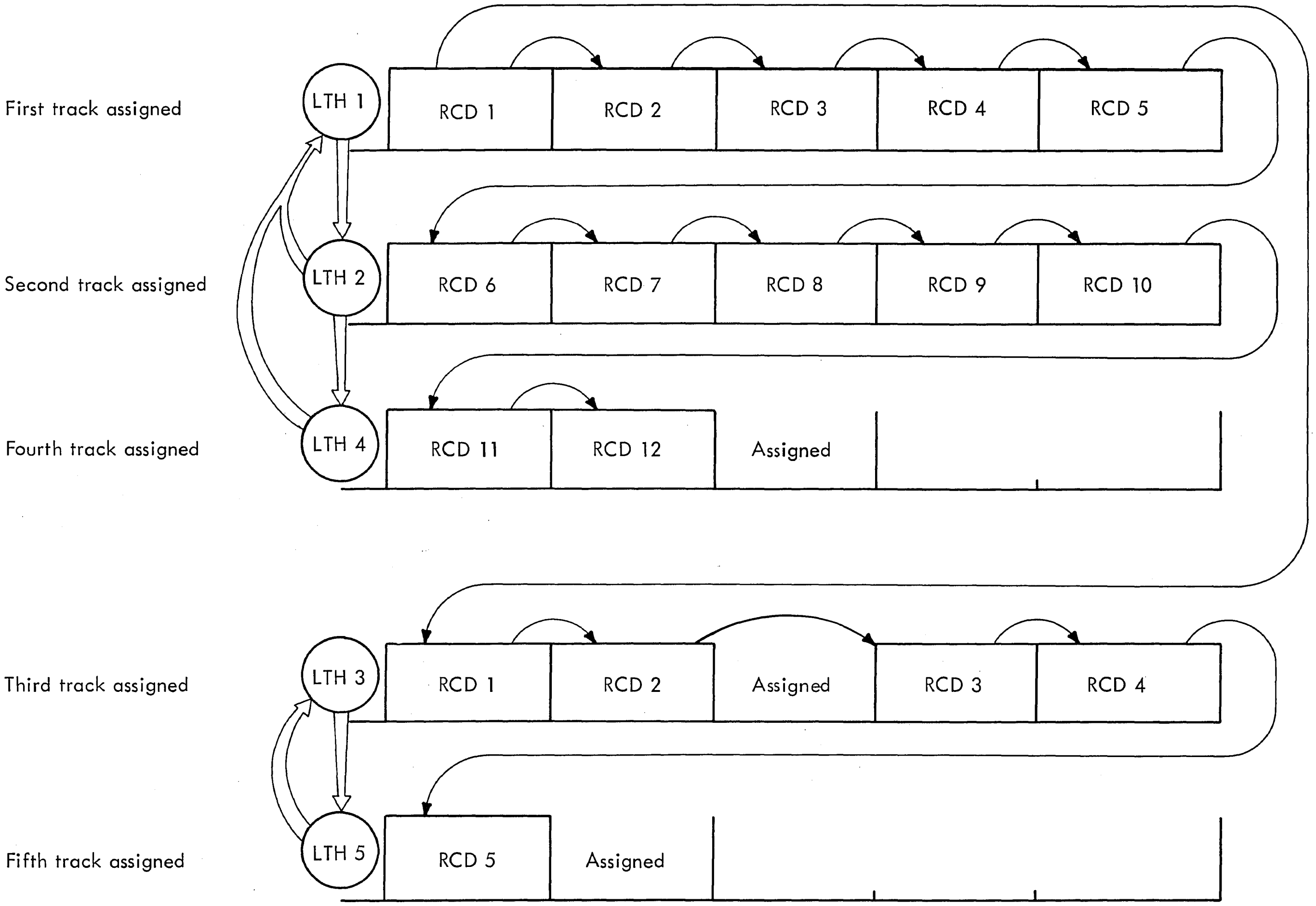


Figure 49. Assignment of Two Sets of Logical Tracks

Figure 50 shows the entry after it has been enqueued. The QCR for the input queue is in a known location in the queue data set; it has been updated to point to the last logical track (in the original set) assigned to the entry. The last logical track assigned to the previous entry of the same priority also points to that track; if there were no other entries of that priority, the last entry of the next higher priority would point to the current entry.

Having used queue management to obtain space, write records, and finally to enqueue the job's input queue entry, the interpreter repeats the process for the next job. If more than one reading task is established in the system, the other interpreters are concurrently performing similar processing.

Concurrently with the creation and enqueueing of input queue entries by interpreters, initiators performing initiating tasks dequeue and process input queue entries, then create output queue entries. When an initiator is ready to process a job, it uses the Queue Management Dequeue routine to remove the highest priority entry from an input queue. When an entry is dequeued, it is removed from the chain of enqueued entries. If, in Figure 50, the "previous" entry were the highest priority entry in the work queue, the top-of-queue pointer in the QCR would point to the last logical track assigned to it. When that entry is dequeued, the top-of-queue pointer is updated to point to the next entry in the chain (the "current" entry in the figure), which becomes the new highest priority entry.

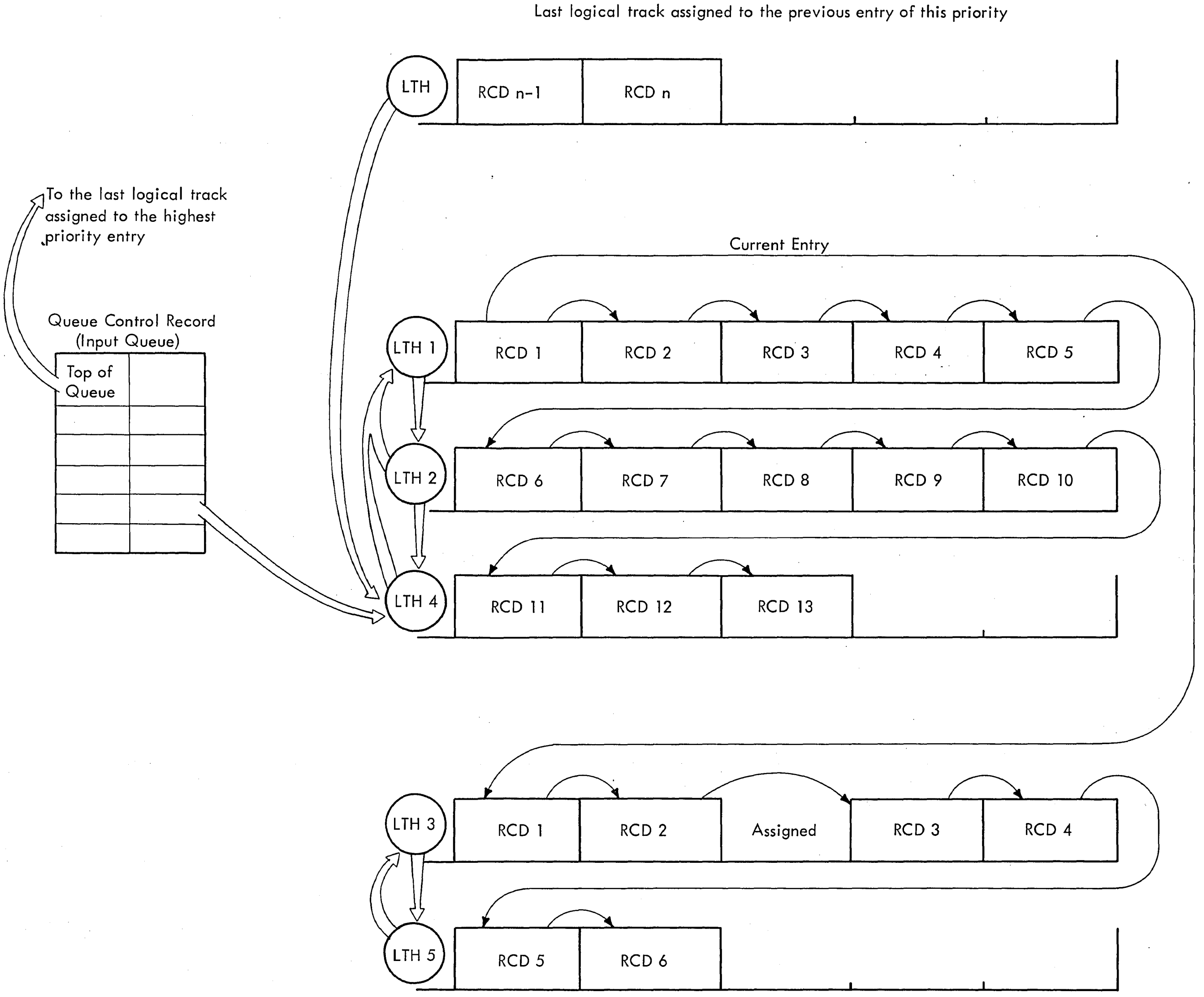


Figure 50. The Entry Enqueued in the Input Queue

As the initiator processes the entry, it uses the Queue Management Read/Write routine to read records from the entry into main storage, and to write the updated records back into their assigned locations. When additional messages to the programmer are generated, routines performing the initiating task use the Queue Management Assign routine to obtain additional record areas (and, if necessary, additional logical tracks). The Queue Management Read/Write routine is used to place the records in the entry.

When the job terminates, the initiator creates records describing the job's system output data sets, and writes them into the previously assigned, but unused, locations in the entry. Finally, the initiator enqueues the set of tracks containing messages and system output data set descriptions in the appropriate output queue, then deletes the input queue entry.

Figure 51 shows the output queue entry after it has been completed and enqueued. Record 7 is a description of a system output data set; it was written (at the request of the initiator) into the area reserved for it by the interpreter. Record 8 contains additional messages to the programmer; the space for it was assigned, and it was written, after the job terminated.

The input queue entry is also shown in Figure 49. Although it is physically present, it will no longer be referred to, except by the Queue Management Delete routine. The Delete routine will make the logical tracks the entry occupies available for assignment to other entries.

When the output queue entry becomes the highest priority entry in the queue, it will be dequeued by a system output writer. After the messages and data set descriptions have been processed, it will be deleted, and its logical tracks will be available for assignment to other entries.

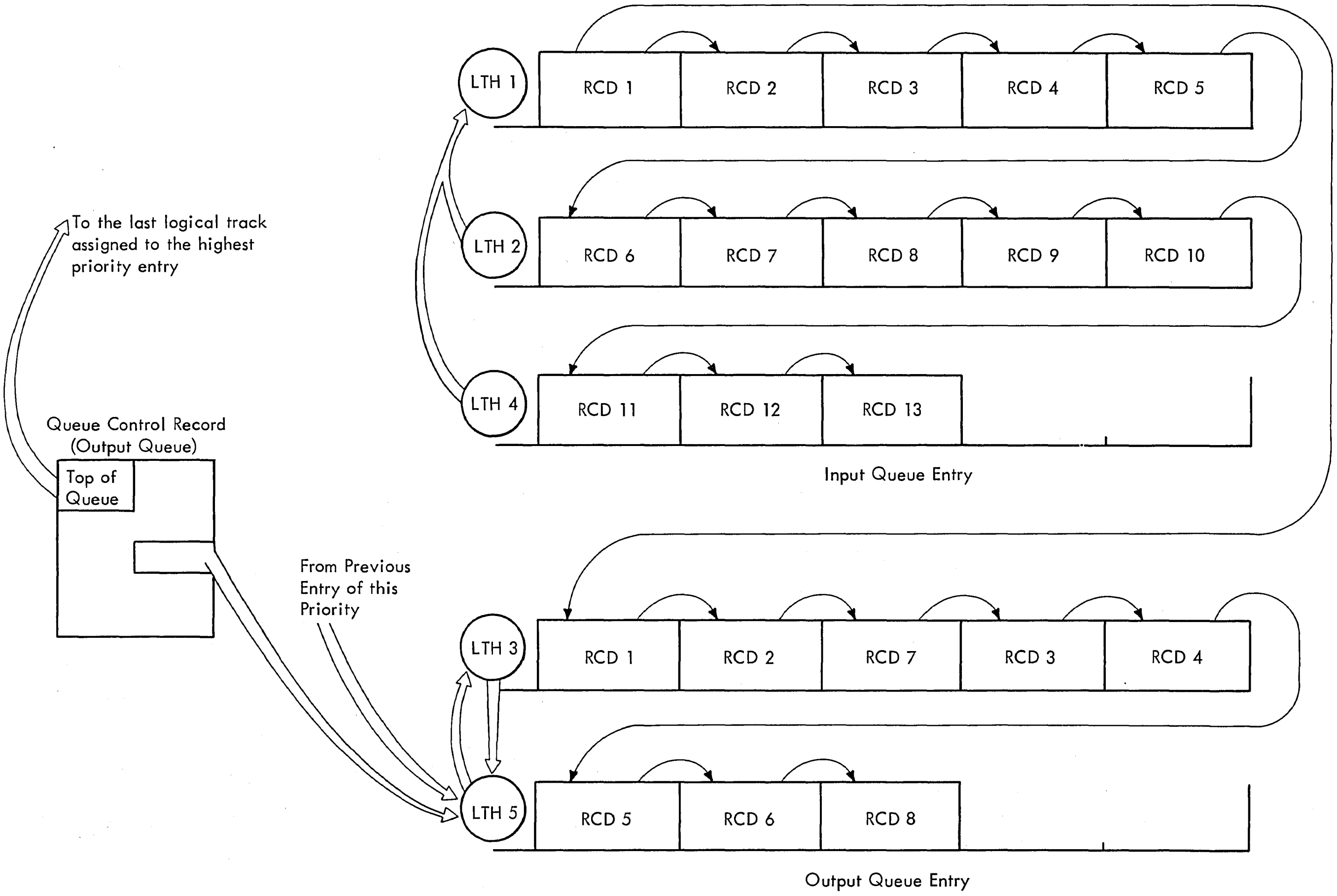


Figure 51. The Entry Enqueued in an Output Queue

Using Queue Management

The routine calling a Queue Management routine must provide a 36-byte queue manager parameter area (Figure 52). In this area, the caller specifies the function to be performed, and includes information about the request, such as queue to which the entry belongs and the number of records to assign, or the addresses of records to read or write. The first 20 bytes of the QMPA correspond to the logical track header (LTH) record. When the Queue Management routines create or update an LTH, they do so in the QMPA, and write the LTH to the queue data set from the QMPA.

Most of the fields in the QMPA are self-explanatory. The bits in the Job Status Code field and the Priority field, however, require further description, and are discussed below:

- In the Job Status Code field, bits 0-1 are reserved. Bits 2-7 have the following meanings when set to one:

Bit	Meaning
2	If a queue full condition is detected during an attempt to assign space, the routine returns to the caller with a return code of 4. If this bit is off, the routine issues a WAIT macro instruction.
3	The input queue entry corresponding to this RJE collector job has been created.
4	The job failed (set only for system output queue entries).
5	The job has been enqueued.
6	The priority has been changed (in the input queue).
7	The job has been canceled (in the input queue).

- In the Priority field, bit 0 is reserved, bit 1 indicates a system task, and bits 4-7 represent the priority assigned to the job. Bits 2 and 3 have the following meanings:

  Bit 2=0 if the caller of the Queue Management routine is an interpreter.

  Bit 2=1 if the caller is an initiator.

  Bit 3=0 if the job's priority has not been changed.

  Bit 3=1 if the job's priority has been changed.

- In the field labeled Address of User's ECB/IOB, the high-order (indicator) byte contains two bits with the following meanings:

  Bit 1 (X'40') = 1 indicates that transient queue management routines are to initialize the ECB/IOB field.

  Bit 2 (X'20') = 1 indicates that transient queue management routines are not to issue a WAIT macro instruction after an EXCP macro instruction has been issued.

When records are assigned, the Queue Management routines update the parameter area with information on the current status of the queue entry.

Unless a queue entry is to be enqueued (and no record is to be written) or deleted, the caller must also provide an additional area, called the external parameter area. If a read or write operation is to be performed, a buffer for the records themselves must also be provided.

Using the ENQ and DEQ Macro Instructions

The Queue Management routines use the ENQ and DEQ macro instructions to prevent concurrent access to a queue control record by routines performing different tasks. There are two queue elements used; one applies to the master QCR, and the other applies to all other QCRs. Each routine that refers to or modifies QCRs issues the ENQ macro instruction (specifying the appropriate queue element) before attempting the access, and issues the DEQ macro instruction when its QCR processing is complete.

Offset

Hex	Dec	Queue Manager Parameter Area			
0	0	Job Name or No-Work-Chain Element 8			
8	8	Function Code 1	NN of First Logical Track Assigned to This Entry 2	No. Records Assigned in This Track 1	
C	12	NN of Next Logical Track 2	No of Logical Tracks Assigned 1	Job Type Code 1	
10	16	Job Status Code 1	Priority 1	NN of Next Queue Entry 2	
14	20	Queue Entry Identification 2	Origin Class ID 1	Reserved 1	
18	24	Address of Track Stacking Parameter List 4			
1C	28	(Indicator) (1)	Address of User's ECB/IOB 4		
20	32	Number of Records to Assign	Number of Records to Read/Write	Address of External Parameter Area 3	
24	36				

Logical Track Header

Figure 52. Queue Manager Parameter Area

Input/Output Operations

In order to perform the function of maintaining the work queues, the Queue Management routines must transfer QCRs, LTHs and 176-byte data records between main storage and the queue data set. Figure 53 shows the sequence of I/O operations performed by the Queue Management routines. QCRs and LTHs are always transferred via XDAP; 176-byte data records are transferred via either XDAP or the Track Stacking routines. During the initial formatting operation (see the section "Initializing the Queue Data Set"), BSAM is used. Since BSAM is discussed in the publications IBM System/360 Operating System: Data Management Services, GC26-3746, and IBM System/360 Operating System: Data Management Macro Instructions, GC26-3794, this section will discuss only the Track Stacking routines.

When 176-byte data records are to be transferred, the caller of the Queue Management routine supplies the record area and TTRs. If track stacking is not specified, the Queue Management routine uses XDAP, and the records are transferred directly between the record area and the queue data set. An access to the queue data set is thus required each time a record is transferred.

If track stacking is specified, the number of accesses to the queue data set is reduced, because 176-byte records are not transferred directly to or from the queue data set. If a record is to be read (or written to a preassigned location), the entire logical track (except for the LTH) is read into one of a set of chained buffers in main storage called a track stack. When the logical track is in main storage, the record is moved to or from the record area. The logical track is kept in the stack as long as possible; it is written out (if any of the records it contains have been updated) only when the buffer it occupies is needed for another track. If the records in a logical track have not been updated, the track is not written out; another track is read in over it if the space is required.

A stack may be shared by routines performing a single system task, even though they may be using several QMPAs. It may not, however, be shared by routines performing separate tasks. For example the initiator, which processes both input and output queue entries, uses one stack and two QMPAs. Each initiator, however, has its own stack.

Queue Management Routine	I/O Operations Performed
Assign	1. Read the first LTH in the free-track queue. 2. Write the newly assigned LTH or (if there is a previously assigned LTH) write the last LTH to be assigned. 3. Write the new LTH. 4. Write the Master QCR.
Enqueue	1. Write the first record of the job (if requested). 2. Read the appropriate QCR. 3. Read the last LTH assigned to the next higher priority entry (or to the last entry enqueued at the same priority). 4. Write the last LTH assigned to the entry being enqueued. 5. Write the last LTH assigned to the next higher priority entry (or to the last entry enqueued at the same priority). 6. Write the QCR.
Dequeue	1. Read the appropriate QCR. 2. Read the LTH of the highest priority job in the queue. 3. Write the updated QCR. 4. Read the first 176-byte record in the dequeued entry (if requested).
Delete	1. Write the last LTH assigned to the entry. 2. Write the last LTH assigned to the free-track queue. 3. Write the master QCR.

Figure 53. Sequence of I/O Operations Performed by Queue Management Routines

Track stacking is performed by three routines, which receive control via a BALR instruction issued in a Queue Management routine when track stacking is specified. The Track Stacking routines are:

- The Stack Initialization routine (module IEFSD110), which obtains main storage for a stack and initializes it.
- The Record Accessing routine (module IEFSD111), which uses EXCP to transfer logical tracks between the queue data set and the stack, and transfers records between the stack and individual record areas.
- The stack purge program (module IEFSD112) uses EXCP to write, from the stack to the queue data set, all logical tracks that have been updated. In addition, it frees the main storage occupied by the stack.

Figure 54 shows the usage of these routines by the Queue Management routines.

Queue Management Routine	Track Stacking Routine Usage
Assign/Start	The Stack Initialization routine builds a stack.
Assign	Whenever the Assign routine must obtain a new logical track from the free-track queue, the Record Accessing routine is used to reserve a buffer and place it at the top of the stack.
Enqueue	The Record Accessing routine may write a 176-byte record. The Stack Purge routine deletes the stack.
Dequeue	The Stack Initialization routine builds a stack. The Record Accessing routine reads the JCT.
Delete	The Stack Purge routine deletes the stack.
Read/Write	The Record Accessing routine reads and writes 176-byte data records.

Figure 54. Usage of Track Stacking Routines by Queue Management

Specifying Track Stacking

When track stacking is to be performed, it is specified by the routine calling a Queue Management routine. The calling routine supplies a track stacking parameter list and places a pointer to it in the QMPA. The parameter list is a 4-byte area; its high-order byte specifies the number of logical track buffers in the stack (n), and its three low-order bytes are reserved for the address of the stack, which is furnished by the Stack Initialization routine when it has created the stack. If there is a pointer in the QMPA, and the high-order byte of the parameter list is non-zero, the queue management routines use the Track Stacking routines. If the calling routine is an initiator, the track stacking parameter list is extended four bytes to include a 1-byte field used by the Assign routine to count the number of tracks allocated to the initiator and a 3-byte pointer to the name of the job being initiated.

Initializing a Stack

The Stack Initializing routine (module IEFSD110) is entered from the queue management Assign/Start or Dequeue routine when the caller has specified track stacking.

If a track stack was passed to the Stack Initializing routine by the caller, the routine bypasses all of the processing described below except the freeing of the user's register save area, and control is returned to the caller.

The Stack Initializing routine calculates the amount of main storage required for the stack (see Figure 55) using the following formula:

Storage required=$88+37N+x+n(8+176N)$

Where: N= the number of 176-byte records per logical track,
n= the number of logical track buffers requested, and
x= an internally calculated value from 0 to 3, used to make the quantity $37N+x$ a multiple of 4.

When the length of the area has been calculated, the Stack Initialization routine issues the GETMAIN macro instruction to obtain the storage from subpool zero; if the TCB indicates that a TSO task is being processed, the routine obtains storage for the stack in subpool 1. It stores the length of the stack in a field 72 bytes from the beginning address of the area, and stores the address of the stack in the track stacking parameter list.

It initializes the channel program and the pointers to the logical first buffer and the first physical buffer. It stores the contents of the 72-byte register save area (which is obtained and passed to it by the Assign/Start or Dequeue routine) in the first 72 bytes of the stack area, then frees the main storage occupied by the register save area, and returns control to the assign/start or dequeue routine.

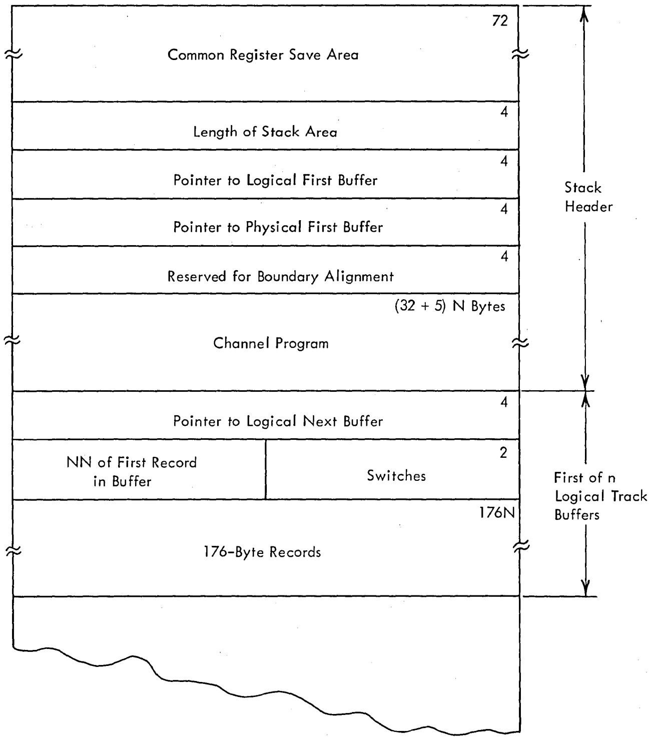

72
Common Register Save Area
4
Length of Stack Area
4
Pointer to Logical First Buffer
4
Pointer to Physical First Buffer
4
Reserved for Boundary Alignment
(32 + 5) N Bytes
Channel Program
Stack Header
4
Pointer to Logical Next Buffer
2
NN of First Record in Buffer
Switches
176N
176-Byte Records
First of n Logical Track Buffers

Figure 55. Track Stack Area

Record Access

The functions of the Record Accessing routine (module IEFSD111) are to move 176-byte data records between the track stack and the queue data set, and when called by the Queue Management Assign routine, to reserve buffers for newly assigned logical tracks. In performing either function, it may be necessary for the routine to transfer logical tracks between the queue data set and the stack.

If the caller has specified track stacking, the Enqueue, Dequeue, and Read/Write routines call the the Record Accessing routine whenever a 176-byte record must be transferred. Also, if the caller of the Assign routine has specified track stacking, the routine will be entered whenever the Assign routine obtains new logical tracks from the free-track queue. The interface between the Record Accessing routine and the Queue Management routines is shown in Figure 56.

When the Record Accessing routine is entered, the QMPA points to a field containing the TTR address of a record. The routine uses the TTR to determine the NN of the record, its relative record number within the logical track, and the NN of the first 176-byte data record in the logical track.

If the relative record number is zero, the record is an LTH, and the Record Accessing routine has been entered from the Assign routine to reserve and initialize a buffer. The routine searches the stack for an unused buffer:

- If it finds an unused buffer, the routine stores the NN of the first 176-byte data record of the track, then updates the buffer chain fields so that the buffer becomes the top buffer in the stack.

- If there is no unused buffer, the routine reserves the buffer at the (logical) bottom of the stack. If the buffer's write indicator is on, the routine writes the contents to the queue data set before initializing. If the buffer's write indicator is off, the routine stores the NN of the first 176-byte data record of the track, then updates the buffer chain fields so that the buffer becomes the top buffer in the stack.

If the relative record number is nonzero, the Record Accessing routine has been entered to transfer a 176-byte data record to or from the stack. The routine searches the stack for a buffer whose identifying NN is the same as the NN of the first 176-byte data record in the logical track that contains the record to be transferred:

- If a buffer with the same NN is found, the chain is updated so that the buffer is at the logical top of the stack, and the record is transferred to or from the buffer according to the code in the QMPA.

- If no buffer with the same NN is found, the logical track must be read into the stack before the record can be transferred. The routine looks for an unused buffer; if it finds one, it reads the logical track into the buffer, updates the chain to make that buffer the logical top of the stack, and transfers the record as specified in the QMPA. If there are no unused buffers, the routine takes the buffer at the logical bottom of the stack. If that buffer's write indicator is on, the routine writes the contents to the queue data set before reading the required track and processing as described above.

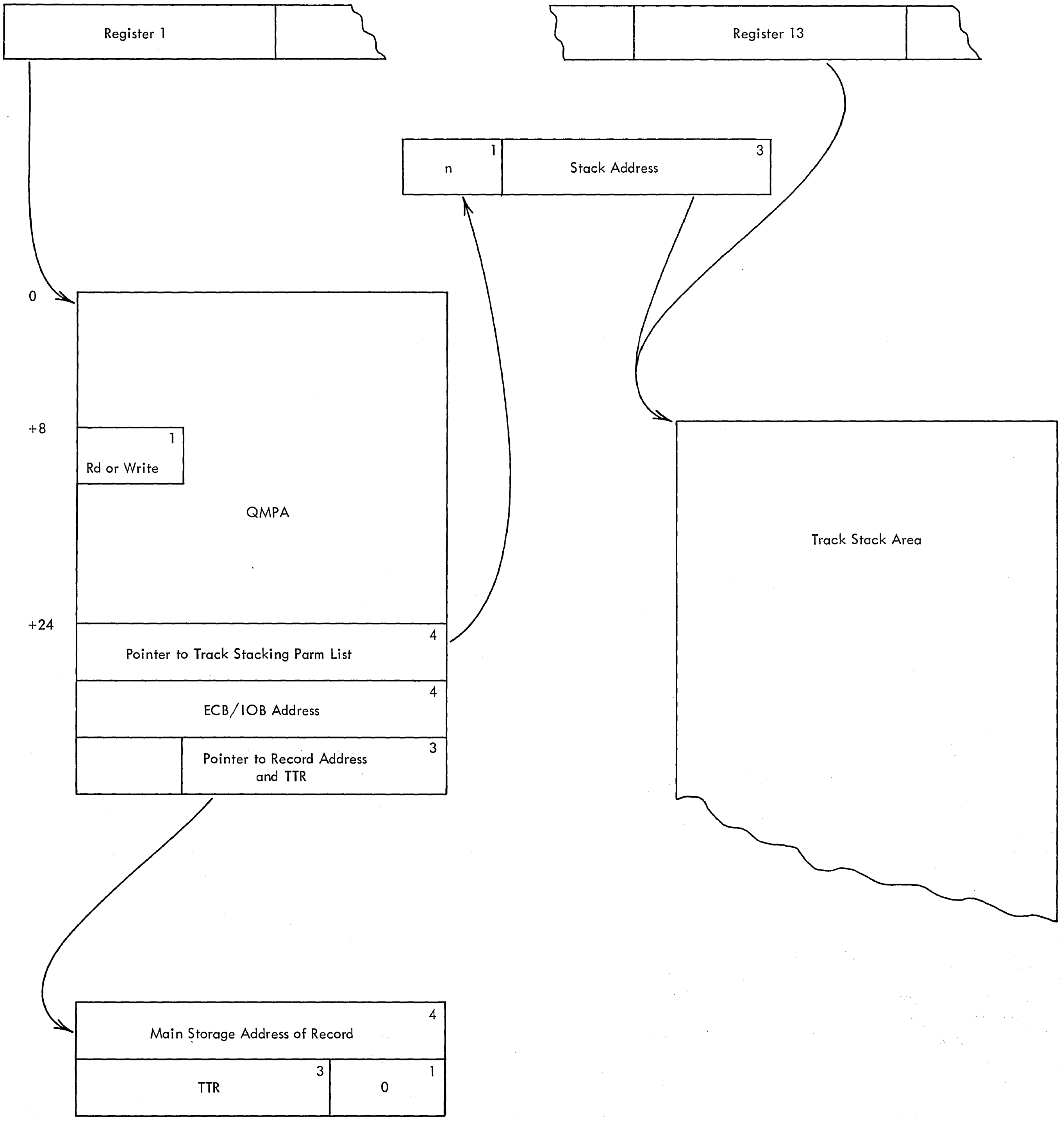


Figure 56. Record Accessing Routine Interface

Whenever data is transferred to a buffer, the Record Accessing routine turns the buffer's write indicator on.

Purging a Stack

When the routines performing a system task complete the processing of a queue entry, when a new region is to be obtained, and when a job step TCB is to be attached, the track stack is no longer required. The Stack Purge routine (module IEFSD112) is therefore entered from the Queue Management Enqueue or Delete routine, and from the initiator region size and Step Attach routines, when the caller has specified track stacking.

If the Stack Purge routine is entered from any routine except the Queue Management Delete routine, it must write out any updated logical tracks still in the stack. It examines the write bits associated with the buffers in the order in which they are chained; all buffers whose write bits are on are written out in order except the first buffer, which is written out last. Thus, although the Track Stacking routines may not place records in the queue data set in the order in which they were updated, the presence of the last record updated (usually the SCT) indicates that all previously updated records are also in the queue data set.

When the updated logical tracks have been written out, or if it is entered from the Queue Management Delete routine, it obtains a 72-byte area in main storage and places the address of the area in register 13. It then moves the first 72 bytes of the stack into the area, frees the main storage occupied by the stack, and returns control to the Queue Management Delete routine.

Assigning Queue Space

Whenever a queue entry is started, or records are to be added to an existing entry, space in the queue data set must be assigned to the entry. A queue entry is started whenever an interpreter encounters a new job in the input stream, and whenever a DD statement is found that specifies a system output class not previously encountered in the current job. Records are added to the queue entry as job control language (JCL) statements are processed by an interpreter; records are also added to queue entries by the I/O Device Allocation routine and the Termination routine.

Assign/Start

It is the function of the Queue Management Assign/Start routine (module IEFQAGST) to prepare for the establishment of a new queue entry. The routine builds an event control block (ECB) and an input/output block (IOB) for use when records are read from or written to the queue, and (unless track stacking is specified) returns control to the caller.

If track stacking is specified, there is a pointer in the QMPA to a track stacking parameter list. The parameter list specifies the number of buffers required for the stack, and contains space for a pointer to the stack area.

If the number of buffers field is not zero, the Assign/Start routine passes control to the Stack Initialization routine (module IEFSD110), which obtains main storage for the stack from subpool 0 or subpool 1 if a TSO task is being processed, initializes it, places the address of the area in the parameter list, and returns control to the Assign/Start routine. The Assign/Start routine then returns control to its caller.

Assign

When the Assign/Start routine has been executed, other queue management functions may take place. It is necessary to have space assigned to the queue entry before records can be written, and it is the function of the Queue Management Assign routine (module IEFQASGQ) to assign space as required.

The primary users of the Assign routine are interpreters. However, since additions may be made to the queue entry by the I/O Device Allocation routine and by the Termination routine, it is possible that lack of available queue space could prevent a job from being initiated or terminated.

To prevent this situation from occurring, two thresholds are established according to parameters submitted when the system is generated, or when the queue is initialized.

The two parameters are the number (T) of 176-byte records to be reserved for each initiator and the number (K) of 176-byte records to be reserved for terminating a job. These parameters are converted to numbers of logical tracks with the formula:

$$\left(\frac{T-1}{N}\right)+2 = \text{number of logical tracks}$$

where N= the number of 176-byte records per logical track.

T is then added to K; the sum is T', which is the number of logical tracks that must be reserved for initiator use in terminating a job.

As each initiator is started, the sum of T and T' is compared to the number of available tracks (the number of tracks in the free-track queue plus the number of tracks actually in use by initiators). If the sum of T and T' is greater than the number of available tracks, the reserve indicator is tested. If a reserve exists, it is used for this initiator. If no tracks have been reserved, the START INIT command is rejected and the operator is notified. When enough space is available for the request to be honored, an attempt is made to add to T' an extra T tracks so that an additional initiator can be started. If this is possible, the reserve indicator is set on.

When any task except an initiating task or an interpreter executing a START command requests the assignment of records to a queue entry, the number of records is converted to logical tracks and subtracted from the number of tracks in the free-track queue. If the result is less than T' minus the number of tracks actually in use by initiators, the Assign routine tests bit 2 of the job status code field in the QMPA. If the bit is on, the routine sets a return code of 4 and returns control to its caller; if the bit is off, the interpreter is placed in the wait state until additional tracks become available.

When an initiator executing a START command requests the assignment of queue space, the Assign routine compares the number of tracks requested to the number of tracks in the free-track queue. If enough space is available, the number of logical tracks requested is subtracted from the number available, and the assignment is made. The count of the number of tracks allocated to the initiator is updated. (This count is the first byte in the second word in the track stacking parameter list.) If there is not enough space available, the Assign routine informs the operator.

When the assignment of logical tracks is requested by an initiator the request is inspected to make sure that honoring it would not increase the number of logical tracks used by the initiator in running this job to more than T. If so, the job is terminated and the operator informed.

When an assignment of logical tracks is requested, and no logical tracks are available, the operator is informed. This situation can result in the system becoming interlocked and unable to continue; in such cases the queue data set must be reformatted, with a larger extent. If so, the operator is informed.

In addition, when an initiator is to be started, T' is recalculated. The number of logical tracks to be reserved for each initiator (T) is added to the old value of T', and if the result is less than the number of tracks in the free-track queue, the initiator cannot be started; the START INIT command is rejected, and the operator is notified.

When an initiator is stopped, and the reserve indicator is on, T' is recalculated by subtracting from it, the number of tracks to be reserved for each initiator (T). If the reserve indicator is off, it is turned on, and the subtraction does not take place.

The Assign routine is entered before records are written; it determines whether there are enough records available in the currently assigned track to accommodate the request for space. If so, it places the addresses of the next available records in the external parameter area, increments the records-assigned field in the parameter area, and returns control to the caller.

If there are not enough records left, an additional logical track must be assigned. The Assign routine issues an ENQ macro instruction to prevent access to the master QCR by routines performing other tasks, then uses the top-of-queue pointer in the master QCR to determine the address of the first logical track in the free-track queue. It reads the LTH into main storage, creates an updated LTH for the last logical track assigned to the entry, and updates pointers as shown in Figure 57 and described below:

- The LTH of the logical track obtained from the free-track queue contains a pointer to the next track in that queue; the pointer is placed in the master QCR.

- The newly assigned logical track is chained to the last logical track assigned to the entry, by placing the NN of the LTH of the newly assigned track in the next-track field of the LTH of the last assigned track.

- The next track field of the LTH of the newly-assigned logical track is zeroed out, indicating that it is the last track assigned.

If track stacking has been specified, the Assign routine uses the Record Accessing routine to reserve a buffer for the newly-assigned track. On the return, or if track stacking was not specified, the Assign routine determines whether there are any 176-byte records to be assigned from the last logical track assigned. If so, the routine places their addresses (TTR) in the external parameter area.

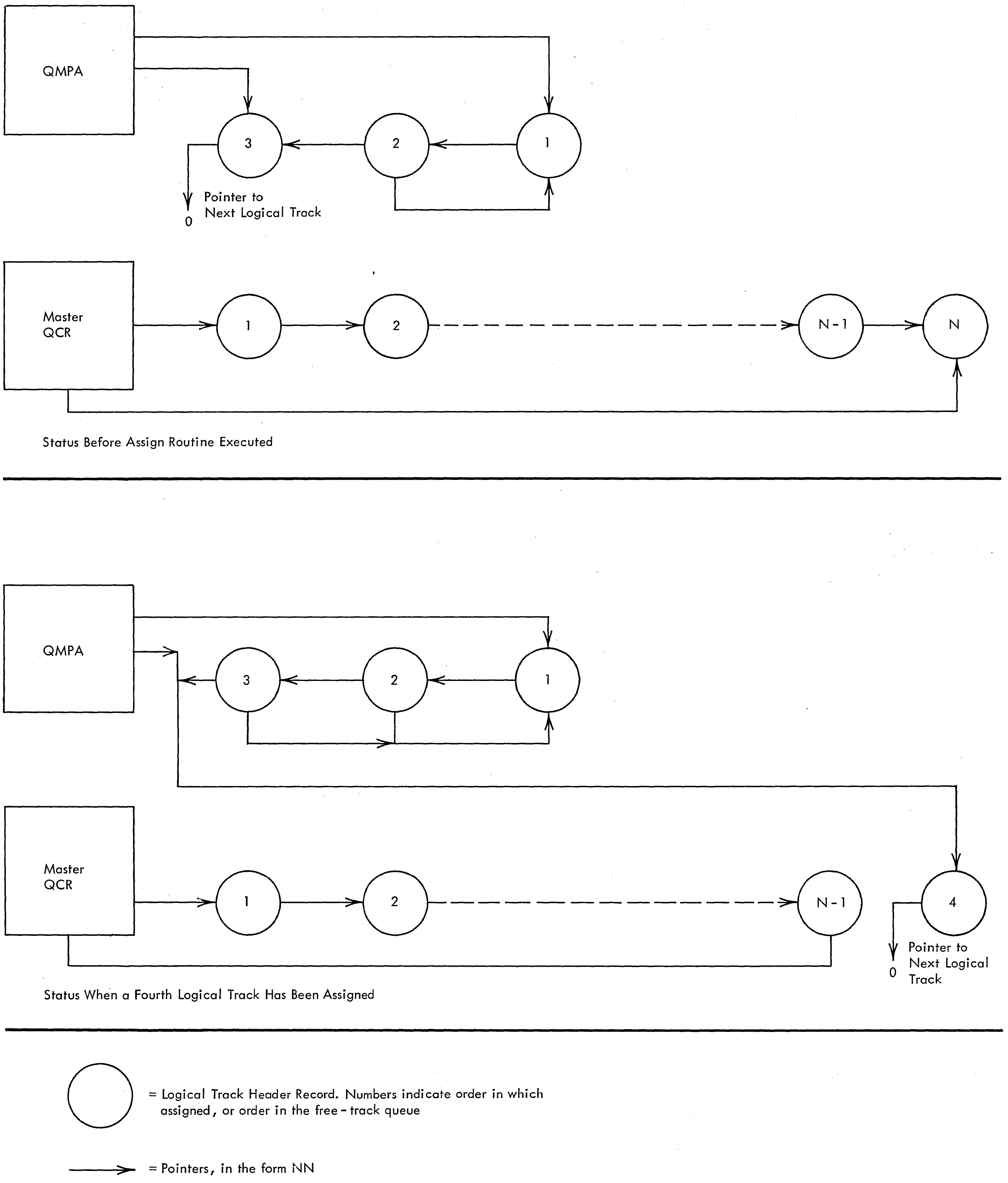

QMPA
3
2
1
Pointer to
Next Logical Track
0
Master
QCR
1
2
N-1
N
Status Before Assign Routine Executed
QMPA
3
2
1
Master
QCR
1
2
N-1
4
Pointer to
Next Logical
Track
0
Status When a Fourth Logical Track Has Been Assigned
= Logical Track Header Record. Numbers indicate order in which assigned, or order in the free-track queue
= Pointers, in the form NN

Figure 57. Logical Track Assignment

The updated LTH for the last-assigned logical track now points to the newly assigned logical track, and the routine writes the LTH to the queue data set. The routine then updates the external parameter area by adding the addresses of the records to be assigned from the newly-assigned logical track. It sets the LTH next logical track pointer to zero, then writes the LTH and master QCR, issues the DEQ macro instruction, and returns control to its caller.

Enqueuing an Entry

A queue entry is built record by record. As JCL statements are processed by an interpreter, they are converted into 176-byte tables; space is assigned, and the tables are written into the queue data set. When interpreter processing for a job is complete, the input queue entry is ready to be enqueued; when the last step has terminated, the output queue entries are ready to be enqueued. Each entry exists as a set of 176-byte records, on one or more logical tracks (which are chained together) in the queue data set. Until it is enqueued, however, an entry is not a member of any queue; the location of the entry is available only in the queue manager parameter area that refers to it, and its priority, relative to that of any other entry, is unknown.

The function of the Queue Management Enqueue routine (module IEFQMNQQ) is to make an entry a part of the appropriate queue, by updating pointers in the queue control record (shown in Figure 58), and in the logical track headers of the entry and of the next higher priority entry.

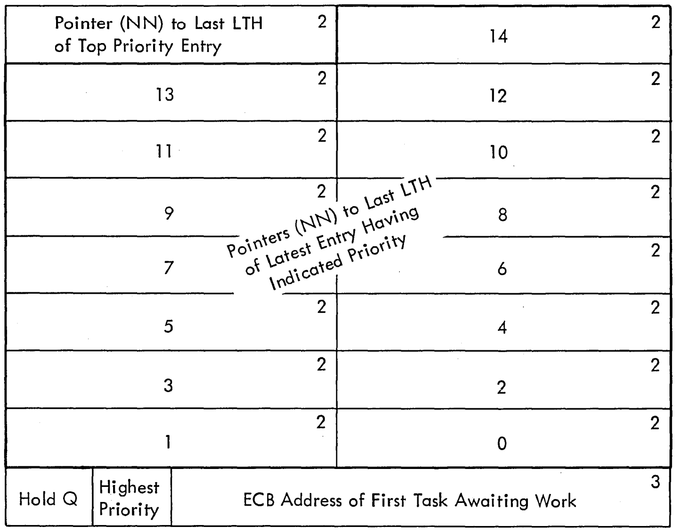

Pointer (NN) to Last LTH of Top Priority Entry
2
14
2
13
2
12
2
11
2
10
2
9
2
8
2
7
6
2
5
2
4
2
3
2
2
2
1
2
0
2
Pointers (NN) to Last LTH of Latest Entry Having Indicated Priority
Hold Q
Highest Priority
ECB Address of First Task Awaiting Work
3

Figure 58. Queue Control Record Format

A typical queue is shown in Figure 59; the QCR points to the last logical track assigned to the highest priority entry in the queue, and to the last logical track assigned to the last entry enqueued under each priority. The LTH of the last logical track assigned to each entry points to the last logical track assigned to a lower priority entry (enqueued later with the same priority, or enqueued with a lower priority).

The Enqueue routine may be used to write a record in the entry before the entry is enqueued; when the interpreter is ready to enqueue an input queue entry, the job control table (JCT) has not been written into the entry (although space has been assigned for it). The interpreter supplies the address of the record in the entry reserved for the JCT, and also its location in main storage. The Enqueue routine places the JCT in the entry, then enqueues the entry.

When it is entered, the Enqueue routine issues an ENQ macro instruction to prevent routines performing other tasks from referring to the QCR.

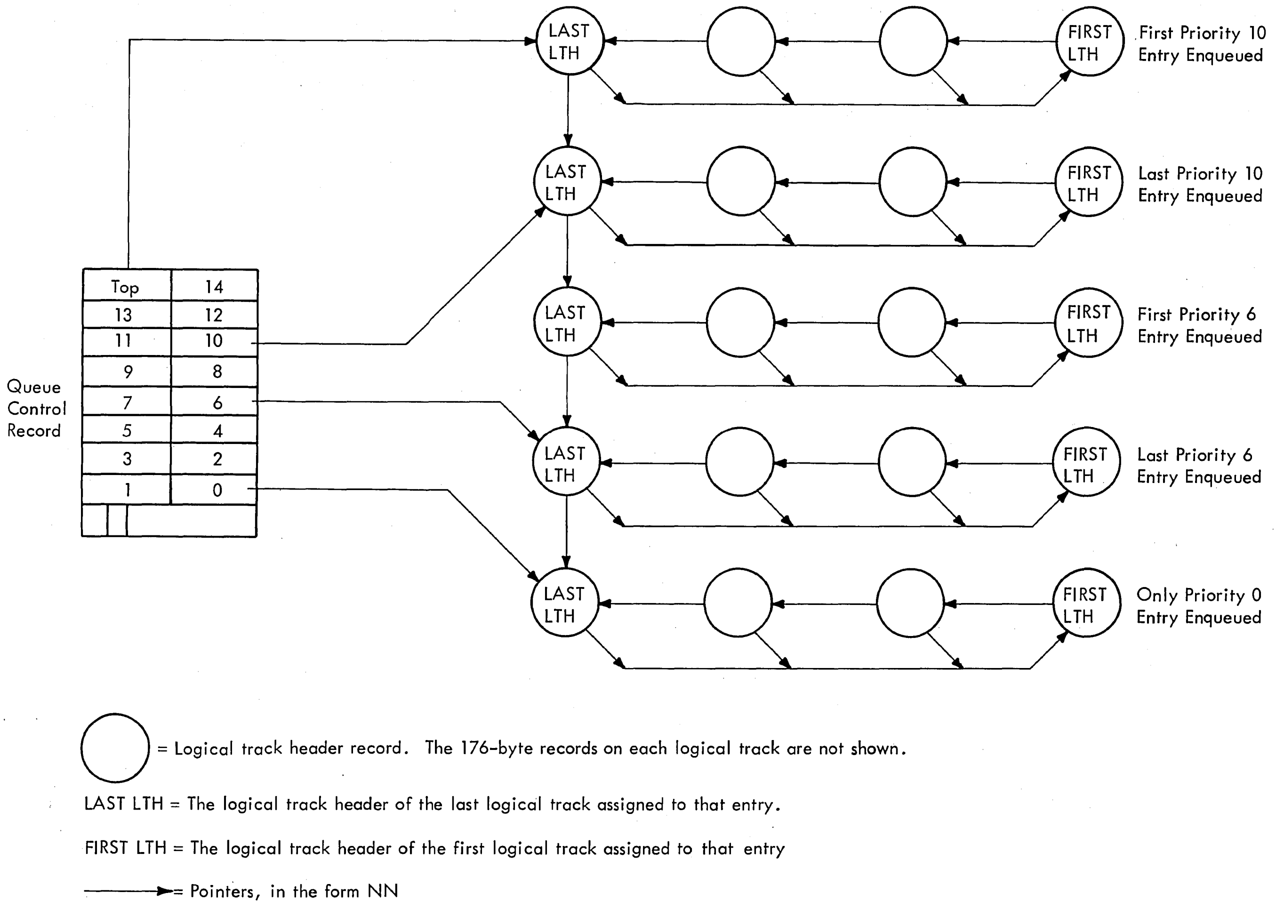
The routine then reads the QCR into main storage, determines from the queue manager parameter area (QMPA) the priority under which the entry is to be enqueued, places the contents of the origin-class-ID field in the function and extracts, from the QMPA, the pointer to the last logical track assigned to the entry.

The QCR bottom-pointer fields are then scanned, starting at the new entry's priority, for a pointer to an entry of equal or higher priority. If a non-zero bottom-pointer field (corresponding to an equal or higher priority) is found, such an entry exists, and the following steps are taken:

- The pointer is the address (NN) of the last logical track assigned to the entry. This address is converted to MBBCCHHR format, and the LTH of that track is read into main storage.

- The LTH contains a pointer to the next queue entry (or zero). This pointer is placed in the LTH for the last logical track assigned to the new entry, and the LTH is written into the queue data set.

- A pointer to the last logical track assigned to the new entry is placed in the LTH read in from the old (equal or higher priority) entry; this LTH is then written back into the queue data set.


LAST LTH
FIRST LTH
First Priority 10 Entry Enqueued
Last Priority 10 Entry Enqueued
First Priority 6 Entry Enqueued
Last Priority 6 Entry Enqueued
Only Priority 0 Entry Enqueued
Queue Control Record
Top
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0

= Logical track header record. The 176-byte records on each logical track are not shown.

LAST LTH = The logical track header of the last logical track assigned to that entry.

FIRST LTH = The logical track header of the first logical track assigned to that entry

= Pointers, in the form NN

Figure 59. Typical Queue

- A pointer to the last logical track assigned to the new queue entry is placed in the QCR bottom pointer field corresponding to the new entry's priority.

If there are no entries in the queue that have a priority equal to or higher than that of the new entry, the following steps are taken:

- The pointer to the next lower priority entry is placed in the LTH of the last logical track assigned to the new entry, and the updated LTH is written into the queue data set.

- A pointer to the last logical track assigned to the new entry is placed in the QCR under the appropriate priority, and in the top-of-queue-pointer field.

- The priority under which the new entry is to be enqueued is placed in the top priority field.

If any tasks are waiting for an entry to be enqueued, the ECB pointer field in the QCR points to the first of a chain of ECBs; the 3 low-order bytes of each ECB point to the next ECB in the chain, and the last ECB is set to zero. Each ECB represents one of the waiting tasks; the Enqueue routine posts each ECB, writes the QCR back into the control record area of the queue data set, issues a DEQ macro instruction, and returns control to the caller.

Dequeuing an Entry

When an initiator is ready to process a job description, and when a system output writer is ready to process a job's system output, they use the Queue Management Dequeue routine (module IEFQMDQQ) to find the highest priority entry in the appropriate queue, remove it from the queue, and read the first record into main storage.

When a queue entry is dequeued, pointers in the QCR for that queue, and pointers in certain LTHs are updated so as to delete the entry from the queue. In addition, the first data record is read into the main storage area specified by the caller of the Dequeue routine, and if the entry is in the input queue, a command scheduling control block (CSCB) is created for the job.

The entry remains in the queue data set (the tracks are not made available for reassignment); records may be read from it, updated, and written back into it. Also, additional queue space may be assigned to it, and records added. The address of the dequeued entry, however, is available only in the queue manager parameter area, and in those data records that contain their own addresses, or pointers to other records.

When the Dequeue routine is entered, it issues an ENQ macro instruction, to prevent routines performing other tasks from referring to the QCR, reads the QCR into main storage, and extracts the top-of-queue pointer.

If the top-of-queue-pointer (which is in NN form) is zero, there are no entries in the queue, and the ECB address field of the QCR is inspected to determine whether there are any other tasks waiting for an entry to be enqueued. If the ECB address field in the QCR is zero, there are none; the address in the first four bytes of the QMPA, which points to an 8-byte element (see Figure 60), is placed in the ECB address field of the QCR. Control is then returned to the caller, with a return code indicating that there are no entries in the queue.

If there are other tasks waiting for an entry to be enqueued, the ECB address field in the QCR contains a pointer to the first element in a "no-work" chain of 8-byte elements, each element being associated with a separate task. The high-order four bytes of each element contain an ECB; the fifth byte contains the name of the queue to which the element applies, and the three low-order bytes contain a pointer to the next element in the chain. If there are other tasks waiting for an entry to be dequeued, the element is added to the chain, then control is returned to the caller.

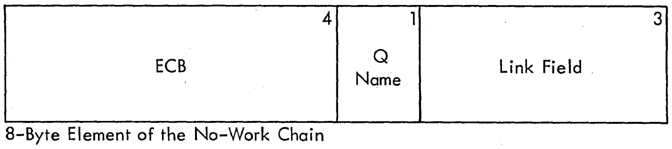


Figure 60. 8-Byte Element of the No-Work Chain

If the top-of-queue-pointer is not zero it is converted to the MBBCCHHR form, and used to read the LTH of the last logical track assigned to the queue entry into main storage.

This LTH contains a pointer to the next queue entry: an entry of the same priority that was enqueued after it, or, if there are no other entries of the same priority, an entry in the next lower priority. If the pointer is zero, there are no more entries in the queue.

If the pointer is not zero, the Dequeue routine determines whether the next entry has the same priority or a lower priority. If the QCR top-of-queue pointer and the bottom pointer (for this entry's priority) are equal, the next entry has a lower priority. The LTH pointer in the dequeued entry is extracted and placed in the QCR top-of-queue-pointer field, the bottom pointer fields are scanned, and when a pointer is found, its priority is placed in the QCR highest priority field.

If, however, the next entry has the same priority, the LTH pointer from the dequeued entry is placed in the top-of-queue-pointer field, but the highest priority field is not changed.

When it has been updated, the QCR is written back into the control record area of the queue data set.

If the entry was dequeued from the input queue, it describes a job; the Dequeue routine constructs a CSCB for the job, and issues SVC 34 to place the CSCB in the CSCB chain, where it remains until the last step of the job has terminated. When SVC 34 has been issued, a DEQ macro instruction is issued.

The LTH of the current entry contains a pointer to the first logical track in the entry; the pointer is converted to the MBBCCHHR form, used to read the first 176-byte record in the entry into main storage, and control is returned to the caller.

Deleting an ECB From the No-Work Chain

When a task must wait for an entry to be enqueued, the WAIT macro instruction specifies an ECB in an 8-byte element (shown in Figure 60). The element is chained, by the Queue Management Dequeue routine, to similar elements created in the performance of other tasks.

If the waiting task is an initiating or writing task, the use of a queue may be terminated as a result of a STOP or MODIFY command. When the command is received, the routines performing the task place the name of the queue in the name field of the element, and pass control (with a pointer to the element) to the Queue Management Unchain routine.

The Queue Management Unchain routine (module IEFQMUNQ) obtains main storage for the QCR, and for an ECB/IOB. It constructs the ECB/IOB, then issues an ENQ macro instruction to prevent routines performing other tasks from referring to the QCRs.

The routine determines whether the ECB in the element has been posted. If it has, a DEQ macro instruction is issued, the main storage is released, and control is returned to the caller.

If the ECB has not been posted, the QCR of the queue specified in the name field of the element is read into main storage. If the element being deleted is the only element in the chain, the Unchain routine sets the ECB address field in the QCR to zero. If not, it deletes the element from the chain.

The routine then writes the QCR back into the queue data set, issues the DEQ macro instruction, and releases the main storage it obtained for the QCR and ECB/IOB, then returns control to the calling routine.

Deleting an Entry

A queue entry remains an entity, and the logical tracks on which it resides remain assigned to the entry, until all processing of the entry is complete. At that time, the initiator or system output writer uses the Queue Management Delete routine (module IEFQDELQ) to return the entry's logical tracks to the free-track queue.

When the Delete routine is entered, it issues an ENQ macro instruction to prevent access by other routines to the master QCR. The routine then extracts the pointer to the last track in the free-track queue from the master QCR.

The LTH of the last track in the free-track queue is constructed in main storage, and is updated with a pointer to the first track formerly assigned to the queue entry. The LTH of the last track formerly assigned to the queue entry is updated to set its next-track-address field to zero (indicating that it is the last track in the free-track queue). The master QCR is updated to point to the new last track and the count of available tracks is incremented. The LTH of the old last track, the LTH of the new last track in the free track queue, and the updated QCR are written into the queue data set.

Finally, the DEQ macro instruction is issued to make the master QCR accessible to routines performing other tasks, and control is returned to the caller.

Transient Queue Management Routines (SVC 90)

The set of Transient Queue Management routines consists of the Transient Queue Management Initialization and Read/Write routine (module IEFXQM00), the Transient Queue Management Track Assignment routine (module IEFXQM01), and the Transient Queue Management Record Assignment routine (module IEFXQM02). These routines perform the same functions as the Queue Management Read/Write routine (module IEFQMRAW), the Queue Management Assign/Start rotuine (module IEFQAGST), and the Queue Management Assign routine (module IEFQASGQ).

The Transient Queue Management routines reside in SYS1.SVCLIB and operate in the transient SVC area. The routines are entered whenever an SVC 90 is issued.

When the Transient Queue Management Initialization and Read/Write routine receives control, it first initializes an ECB/IOB and prepares the QMPA. If the Transient Queue Manager is requested to provide a track or record, the routine passes control via an XCTL macro instruction to the Transient Queue Management Track Assignment routine or the Transient Queue Management Record Assignement routine. If the Transient Queue Manager is requested to read or write a record onto the job queue, the Transient Queue Management Initialization and Read/Write routine performs the read or write function. The routine returns control to the caller upon completion, as does the Transient Queue Management Track Assignment routine and the Transient Queue Management Record Assignment routine.

Dequeue by Jobname Routine (IEFLOCDQ)

The Dequeue by Jobname routine is a queue management function that the RJE, TSO, and Checkpoint Restart modules use for searching a jobqueue for jobs. The routine uses, as subroutines, the modules IEFRDWRT (for reading and writing the queue records) and IEFCNVRT (for converting relative track addresses to absolute addresses).

As queues are searched, module IEFLOCDQ puts into a parameter area, which a user has passed to it, information related to the dequeueing or cancelling of jobs. The particular entry point at which module IEFLOCDQ gains control determines whether the module will locate, locate and cancel, or locate and dequeue a job.

The caller that requests the dequeue routine must provide a parameter list with pointers to the following items:

1. A jobname list.
2. The number of jobs on the list.
3. The queue control record (QCR) number for the jobqueue (i.e. indicating whether the queue is an ASB reader queue, an RJE queue, an input queue, an output queue, etc.).
4. The number of actual jobs found on the queue.
5. An information area in which module IEFLOCDQ will place the following items (for each job located on the queue) to be returned to the caller:

   - The position of the job on the jobqueue.
   - The priority of the job.
   - The status of the job.
   - The relative address of the job on the jobqueue.
6. For jobs to be dequeued, the address of a list of queue entries and the address of an ECB/IOB.

The Interpreter Routine

The information furnished to the operating system in an input stream is in the form of 80-character JCL statements. This information must be converted to a series of standard tables, so that it can conveniently be processed by other job management routines. The conversion is performed by the interpreter routine, used as a subroutine by the other job management routines.

The interpreter (Chart 02) is called by passing control to module IEFVH1, the Interpreter Initialization routine (Chart 03). The interpreter is capable of taking information from an input stream and the procedure library, processing it, and storing it for convenient retrieval by programs executing other tasks. It returns control to its caller when the end of the input stream has been reached, or when an appropriate STOP command has been issued.

An input stream is a sequential data set composed of job control language (JCL) statements, operator command statements, and system input data. Command statements cause the interpreter to issue SVC 34 (see the Command Processing section in this publication) so that the Command Scheduling routine is executed. System input data is written, unchanged, to a direct access device for later retrieval by its associated job step.

The procedure library is a partitioned data set, which may be blocked or unblocked according to the user's specification. Each member (called a cataloged procedure) is a set of JCL statements describing frequently executed series of job steps.

Like a cataloged procedure, an in-stream procedure is a set of JCL statements describing frequently executed series of job steps. However, an in-stream procedure is included in the input stream of a job rather than in the procedure library. As many as fifteen in-stream procedures may be included in one job, and an in-stream procedure may be executed any number of times during the job, to a maximum of 255 job steps.

THE INTERPRETER INTERFACE

Figure 61 shows the data flow and the control and data interfaces of the interpreter. The caller must provide an interpreter entrance list (NEL), and an option list, and he may provide an exit list. These lists are shown in Figures 62, 63, and 64, respectively, and their fields are described below.

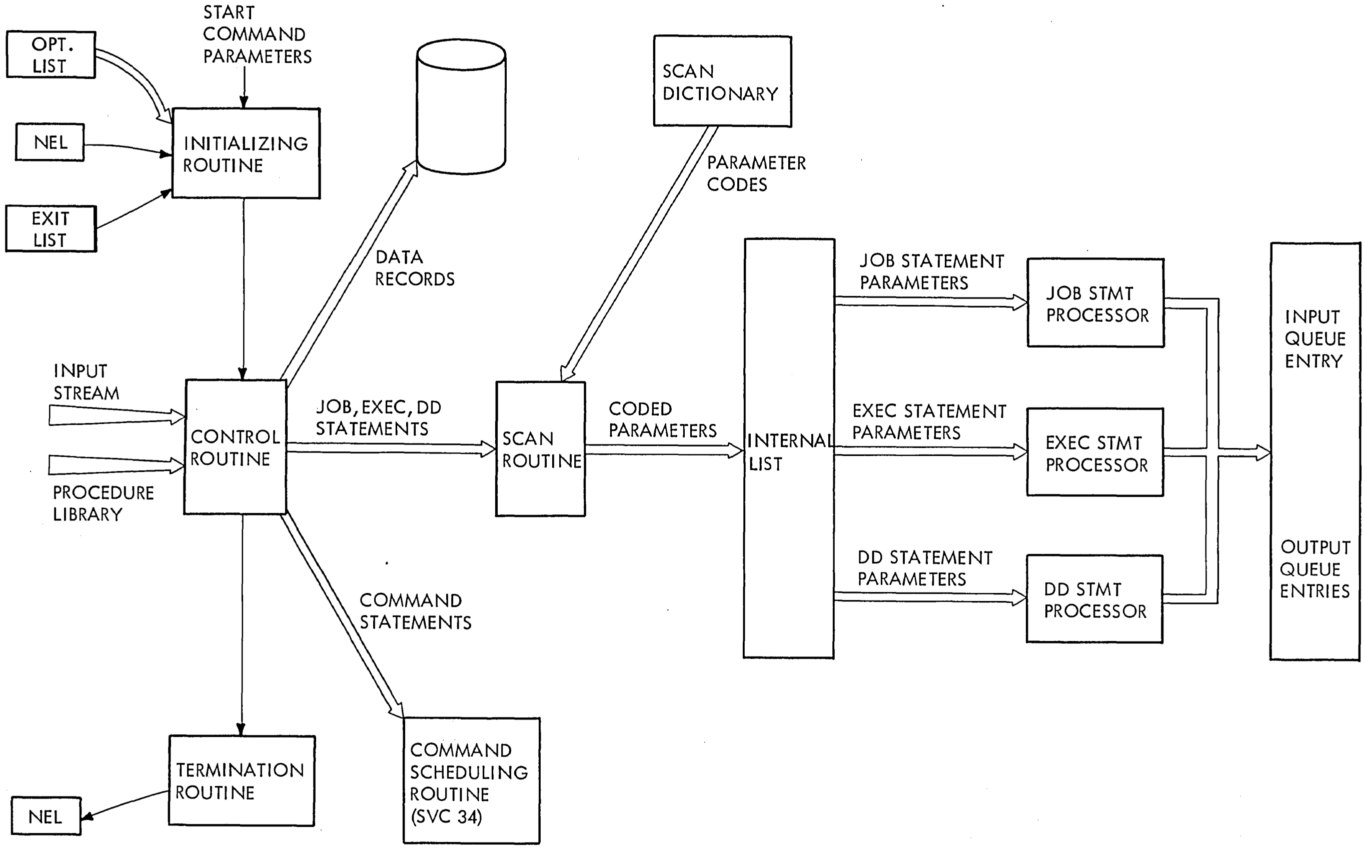


Figure 61. Interpreter Data Flow

The Interpreter Entrance List (NEL)

The interpreter entrance list is a six- or seven-word parameter list that contains pointers to the tables used in communicating information between the caller and the interpreter:

Word	Contents
1	The first word of the NEL contains a pointer to the option list. This pointer, which must be supplied by the caller, is maintained by the interpreter.
2	The second word must contain the address of the STOP ECB.
3	The third word may contain the address of the input stream if it is to be processed by a special access method. If no special access method is to be used, this field is ignored.
4	The fourth word may contain the address of a queue manager parameter area (QMPA). If so, the set of tables created for the first job to be processed by the interpreter will be added to the input queue entry associated with the QMPA. If this field is zero, a separate input queue entry is created for each job. Upon the return to the caller, this field points to the QMPA for the last job processed, unless the enqueue option was selected. In this case, the field is zero.
5	The fifth word is unused upon entry to the interpreter. On the return to the caller, however, it contains a pointer to the JCT constructed for the last job processed by the interpreter, unless the enqueue option was selected. In that case, it will be zero.
6	The sixth word may contain a pointer to the exit list (see Figure 64). If this pointer is supplied, it is maintained by the interpreter; if it is zero, the default exit options apply. If the user omits the optional seventh word from the NEL, he must set the high order bit of the sixth word to indicate that it is the last word in the list.

7 The seventh word is optional and may contain the address of the console identifier, which indicates the console on which operator messages will be displayed. The ID specifies the console that issued the last operator command affecting the interpreter. If the user specifies zeros in the seventh word, the address defaults to the master console. If the user omits the seventh word from the NEL, operator messages will be displayed on the master console. If the user includes the seventh word, he must set the high order bit of the word to indicate that it is the last word in the list.

Option List Pointer	4
ECB Pointer	4
JCL Pointer	4
QMPA Pointer	4
JCT Pointer	4
Exit List Pointer	4
Console ID Pointer	4

Figure 62. Interpreter Entrance List (NEL)

The Option List

The option list, shown in Figure 63, specifies the processing options of the interpreter and the values to be used when default options are taken in job control language statements. It contains the following fields.

- List Length: This field contains a binary number that represents the number of bytes in the option list.

- Option Switches: This field specifies the following interpreter options.

Bit	Explanation
0	If this bit is on (set to 1), the interpreter performs System Management Facilities (SMF) functions.
1	If this bit is on, the interpreter processes the TERM=TS DD keyword.
2	If this bit is on, the SMF options to be used are found in byte NELSMFOP of the NEL.
3	If this bit is on, the interpreter sets bit 2 in the JCT to indicate that all messages for this job are to be suppressed.
4	If this bit is on, the interpreter initiates track stack processing.
5	If this bit is on and the job queue is full, the interpreter deletes all queue entries for the current job, sets a return code of 12, and terminates normally.
6	If this bit is set to 1, the interpreter is not to convert option list fields from EBCDIC to binary.
7	If this bit is on, the interpreter is to enqueue jobs; if the bit is off, the interpreter is to return to the caller a pointer to the JCT constructed for the most recent job, and a pointer to the QMPA associated with the queue entry, but it is not to enqueue jobs.

- System Code: This field specifies the level (MFT or MVT) of the operating system:

Value	System
X'01'	MVT
X'02'	MFT

- SMF Override: This field contains a user-override for standard SMF options.

- Queue Tracks: This field indicates the number of track stacking buffers used by the interpreter.

- Interpreter Number: This field contains a binary number used by the interpreter in the generation of unique data set names. The number should not duplicate any other interpreter number issued since the last IPL procedure was executed.

- Parameter Options: This field contains an EBCDIC character from 0 through F that is the hexadecimal equivalent of the binary value of a four-bit field:

Value	Explanation
X'01'	Step can cause rollout.
X'02'	Step can be rolled out.
X'04'	Fail all jobs that do not specify an account number.
X'08'	Fail all jobs that do not specify programmer's name.

- Default/Priority: This field contains the priority value that the interpreter is to assign when priority is not specified in the JOB statement.

- Default Time Limit: This field contains the time limit the interpreter is to assign when time limit is not specified for a job step.

- Default SYSOUT Primary Quantity: This field contains a value representing the number of tracks to be assigned to system output data sets when no other specification is made.

- Default SYSOUT Secondary Quantity: This field contains a value representing the secondary quantity of tracks to be assigned to system output data sets when no other specification is made.

- Interpreter Dispatching Priority: This field contains a value representing the dispatching priority assigned to the task being performed by the interpreter.

- Default Region Size: This field contains the size of the region that will be requested for a step when no region size specification is made in the EXEC statement.

- Command Authorization: This field specifies the disposition of commands encountered in an input stream:

Value	Disposition
X'80'	Execute the command.
X'40'	Display the command (via a WTO macro instruction), and execute it.
X'20'	Display the command (via a WTOR macro instruction), but do not execute it unless advised to do so by the operator.
X'10'	Ignore the command.

- Label Processing: This field specifies the processing to be performed as a result of the Bypass label processing (BLP) parameter:

Value	Processing
X'80'	Treat BLP as NL.
X'40'	Treat BLP as bypass label.
X'20'	Operator specifies processing action.

- Default SYSOUT Device Name: This field contains the three-character name of the unit to be used for system output data sets if no UNIT parameter was specified in a system output DD statement.

- MCS Command Authority: This field contains a four-byte hexadecimal number that indicates the command groups that may be submitted on input command statements. The command groups are organized as follows:

Group Number	Commands
0	BRDCST, MSG, DISPLAY, SHOW, LOG, REPLY.
1	CANCEL, RESET, CENOUT, SET, DEFINE, START, HALT, STOP, HOLD, USERID, MODIFY, RELEASE, WRITELOG, QUIESCE (Model 65), MODE (Model 85).
2	MOUNT, SWAP, VARY (Nonconsole devices), UNLOAD.
3	VARY (Console devices).
4-15	Reserved.

Group 0 commands are always executed; therefore, they are given no explicit authorization in this field. The four-byte hexadecimal number in the field indicates the other groups that are authorized for execution. The hexadecimal value is derived as follows:

1. The bits in two bytes are numbered one through sixteen.

2. The bits corresponding to the groups authorized for execution are set to 1; all remaining bits are set to 0.

3. The binary value of each of the two bytes is converted to hexadecimal, and the two hexadecimal values are placed in the four-byte field.

- Default JCL Message Level: This field specifies the default for the JCL message level for the MSGLEVEL of the job:

Value	Output
0	JOB statement only.
1	Input JCL statements, cataloged procedure statements, and symbolic parameter substitution messages.
2	Input JCL statements only.

- Default Allocation Message Level: This field specifies the default for the allocation level for the MSGLEVEL of the job:

Value	Output
0	No output: allocation/termination messages are suppressed unless the job step terminates abnormally.
1	Allocation/termination messages.

Note: The allocation message level for system tasks is always zero.

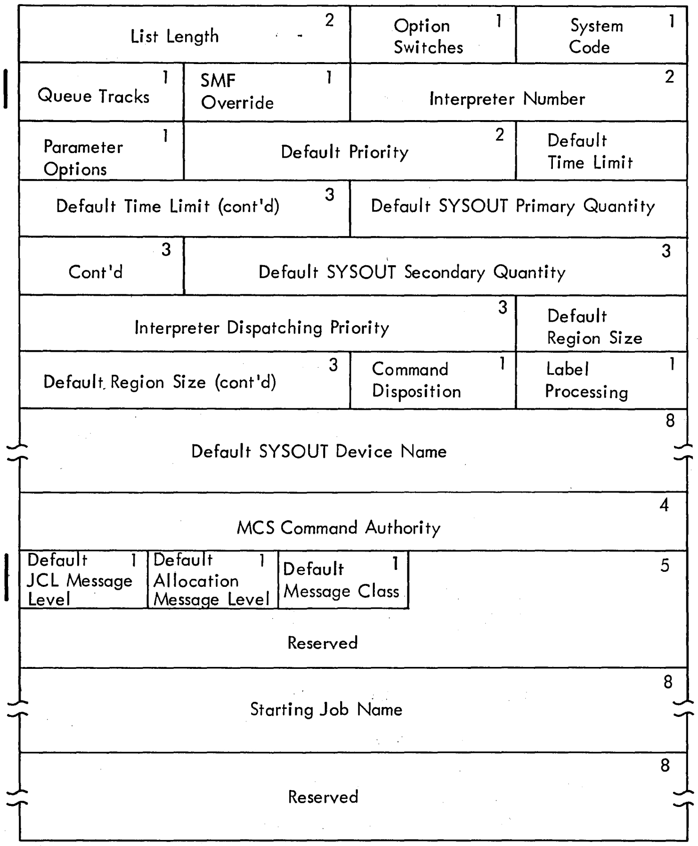


Figure 63. Interpreter Option List

- Default Message Class: This field specifies the default message class. If the contents of this byte are zeros, the default message class is message class A. If nonzero, the contents of this field are used as the default message class.

- Starting Job Name: This field contains either zeros or a job name. If the field contains zeros, the interpreter will start processing jobs with the first job in the input stream; if it contains a name, the interpreter will ignore all jobs in the input stream until the named job is encountered.

The Exit List

The exit list (Figure 64) permits linkage to Optional Exit routines. The list consists of an 8-byte header entry containing the length of the list, and a series of 8-byte data entries, each of which describes an exit. Data entries contain the following fields:

- Exit Definition: This field specifies the type of linkage to be used between the interpreter and the Exit routine (Figure 78, which appears in the Interpreter Exits section, contains a description of linkage types associated with exit identifiers). Bits 0 and 1 contain the specification; bits 2-7 are reserved.

Setting	Type
X'00'	Default
X'40'	Address
X'80'	Name
X'C0'	V-type address constant

- Exit Identifier: Bits 0-7 of this field specify an exit ID which corresponds to the exit specified by this list entry:

ID	Bit	Exit
X'80'	0	Accounting routine
X'40'	1	Input Access Method routine
X'20'	2	Return to caller
X'10'	3	Find Access Method routine
X'08'	4	Queue Manager routine
X'07'	5-7	System Management Facilities Exit routine

- Linkage information: The type and format of the linkage information depends on the specification in the exit definition field:

Specification	Linkage Information
Default	Zeros
Address	3-bytes address, right justified and padded with zeros
Name	6-byte EBCDIC name
V-type address constant	Zeros

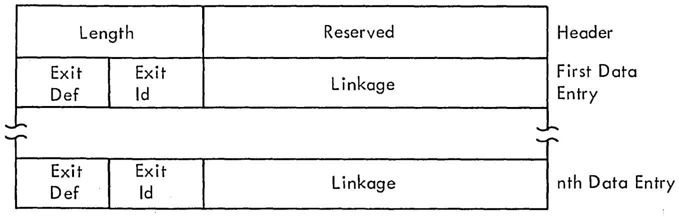


Figure 64. Interpreter Exit List

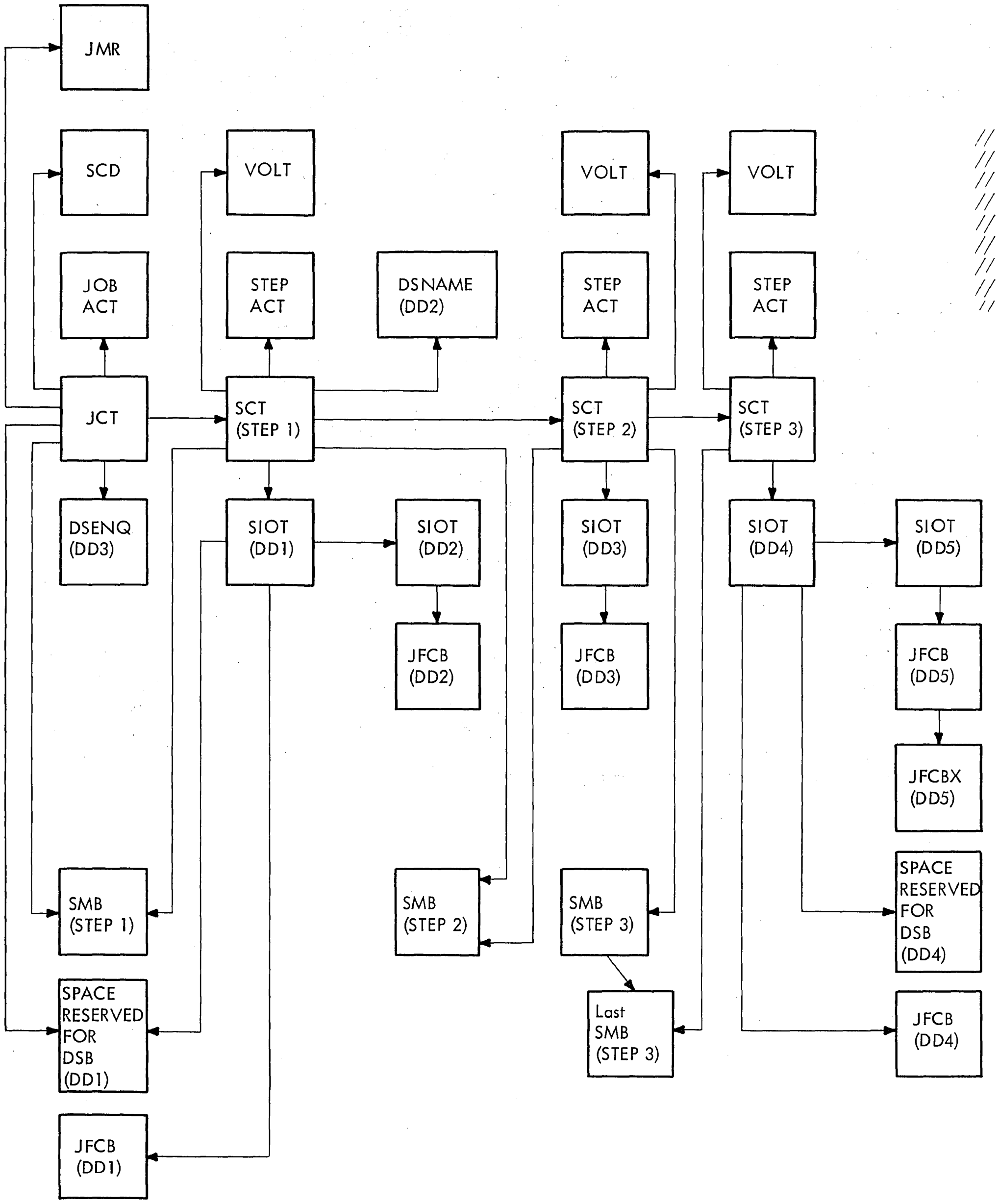
When the lists have been constructed, the caller passes control to the Interpreter Initialization routine (module IEFVH1), which stores the initializing parameters and passes control to the Interpreter Initialization Open routine (module IEFVH2). The routine opens the input stream and procedure library data sets, then passes control to the Control routine (see Chart 04).

The Control routine reads the input stream and procedure library records. It determines the record type, processes command statements and data records, and passes JCL statements to the JCL Scan routine.

The Scan routine (module IEFVFA) converts JCL statements into an internal text format. Since input stream JCL statements may invoke and modify cataloged procedures, the Scan routine accumulates a complete logical statement (which may include several records from the input stream and the procedure library) before further processing is performed. When it has converted the complete logical statement into internal text, it passes the text to the appropriate JCL Statement Processor routine.

The Processor routines (see Chart 06) build the tables for the job's input queue entry, and write them into the queue data set. In addition, they create system message blocks as required, write them into the message class queue entry, and obtain space for the data set blocks (DSBs) required for the job.

The interpreter output for a job is shown in Figure 65. If the output is not to be enqueued in the input queue, the JCT shown in the figure is written out, but no queue entry is made; control is returned to the caller with the NEL containing pointers to the QMPA and the JCT. In either case, the records destined for the output queue entries are enqueued when the job is terminated.



```
// JOBNAME    JOB
// STEP1      EXEC
// DD1        DD SYSOUT=A
// DD2        DD VOLUME=REF=name
// STEP2      EXEC
// DD3        DD DSNAME=NAME
// STEP3      EXEC
// DD4        DD SYSOUT=B
// DD5        DD VOLUME=(,,,17)
```

Figure 65. Typical Interpreter Output (Queue Data Set)

INITIALIZING THE INTERPRETER

The interpreter is entered from the System Task Control routine, either during the initiation of a reading, writing or MOUNT command task, or to perform the reading task.

When the interpreter is entered during the initiation of one of the system tasks, the input stream it is to process is an internal data set created in the System Task Control routine; the interpreter entrance list contains a pointer to the input stream, and instructions for the interpreter that cause it to load a special access method and prevent it from enqueuing the job.

When the interpreter is entered to perform the reading task, the input stream is on external storage; the interpreter TIOT, constructed by the I/O Device Allocation routine, points to the input stream data set, and the interpreter entrance list instructs the interpreter to enqueue jobs.

In either case, the interpreter is entered (via a LINK macro instruction) at the Initialization routine, which consists of two modules: the initialization module (IEFVH1) and the open module (IEFVH2). At entry, control passes to the initialization module, which obtains main storage for the interpreter work area (IWA), and for the local work area (LWA). The IWA contains information that is shared by two or more of the Interpreter routines. The local work area is mapped individually by each major routine (although in some cases two or more routines use the same mapping macro instruction to produce a DSECT of the LWA). The LWA contains only information that need not be preserved outside that routine. Throughout the performance of the reading task by the interpreter, register 12 contains a pointer to the IWA; the IWA contains a pointer to the local work area.

When the main storage for the IWA and local work area has been obtained, the Initialization routine obtains main storage for the data set enqueue (DSENQ) table, the input stream DCB, and the procedure library DCB, and if SMF is included in the system, the job management record (JMR). The routine then stores pointers to these areas in the IWA. A separate storage request is made for the DSENQ table and for the DCBs.

The routine then issues a TTIMER macro instruction and combines the time with the reader number in the CSCB to create the base for any unique data set names to be generated for this input stream. Next, it examines the PARM field passed to it by its caller. It extracts the option fields, and sets the corresponding switches and values in the IWA.

The date and time used in creating temporary data set names is obtained during this initialization process and reside in the IWA throughout the life of the reader. All temporary data set names created by a reader during the interpretation process use the same date and time, both of which never change throughout the life of the reader or the life of the job step in the system.

If SMF is included in the system, module IEFVH1 uses the LOAD macro instruction to bring module IEFUJV into main storage. (IEFUJV is the name that must be assigned to the user's JCL Validation routine; if such a routine has been supplied, it is brought into main storage.) If the user has not supplied a routine named IEFUJV, an IBM-supplied routine, which executes a return to its caller, is loaded.

When module IEFUJV has been loaded, the Interpreter Initialization routine stores, in the JMR, the entry point address of module IEFUJV and the class and type of device allocated to the reading task.

The initialization module finally passes control to the open module, which opens the input stream and procedure library data sets. If the input stream is an internal data set, the third word of the interpreter entrance list contains a pointer to the data set and the entrance list specifies that a special access method is to be loaded. If no special access method is to be loaded, the data set is opened for QSAM. In either case, the procedure library is opened for BPAM.

INPUT AND CONTROL OPERATIONS

When initialization is complete, control is passed to the Interpreter Control routine (Chart 04), which reads records from the input stream and procedure library, determines the record type and processing required, and either performs the processing, or passes control to the appropriate Processing routine.

Reading Control Statements

The Interpreter Control routine is entered at the Interpreter Get routine (module IEFVHA). This routine uses the GET and READ macro instructions to obtain records from the input stream and procedure library, respectively. Only one input source is read upon each entry to the routine; switches set in the Verb Identification routine (module IEFVHCB) determine which data set is read.

When the record is in main storage, the Get routine determines whether it is a control record (// in positions 1 and 2) or a data record. A data record is found only when a data set was included in the input stream, but not preceded by a DD* statement (data records are normally read by the Spool routine). Such a record causes control to be passed to the Sysin DD* Generator routine (module IEFVHB), which builds the missing statement, places "SYSIN" in the name field, and passes control to the Continuation Check routine.

<u>End-of-Data and Null Statement Processing</u>

The physical end of an input stream is signalled by an end-of-data indication from the computing system. A null statement is the last statement in an input stream (or a job description), and is also the last statement in a procedure. Figure 66 shows the processing performed when an end-of-data condition or a null statement is encountered.

An end-of-data condition causes control to be passed to the interpreter EODAD Exit routine (module IEFVHAA), which determines whether there is a job to be enqueued. If not, it passes control to the Interpreter Termination routine (module IEFVHN); if so, it constructs a null statement, and passes control to the Continuation Check routine.

The Continuation Check routine (module IEFVHC) passes control to the Verb Identification routine (module IEFVHCB), which determines that the statement is a null statement, and passes control to the Null Statement routine.

The Null Statement routine (module IEFVHL) is given control by the Verb Identification routine whenever it encounters a null statement. The Null Statement routine examines the conditions under which it was entered, and passes control as described here. The circled numbers refer to Figure 66.

① If the null statement represents the end of an input stream job, and more records must be processed from a procedure but have not yet been read, control is passed to the Interpreter Get routine so that reading can be continued.

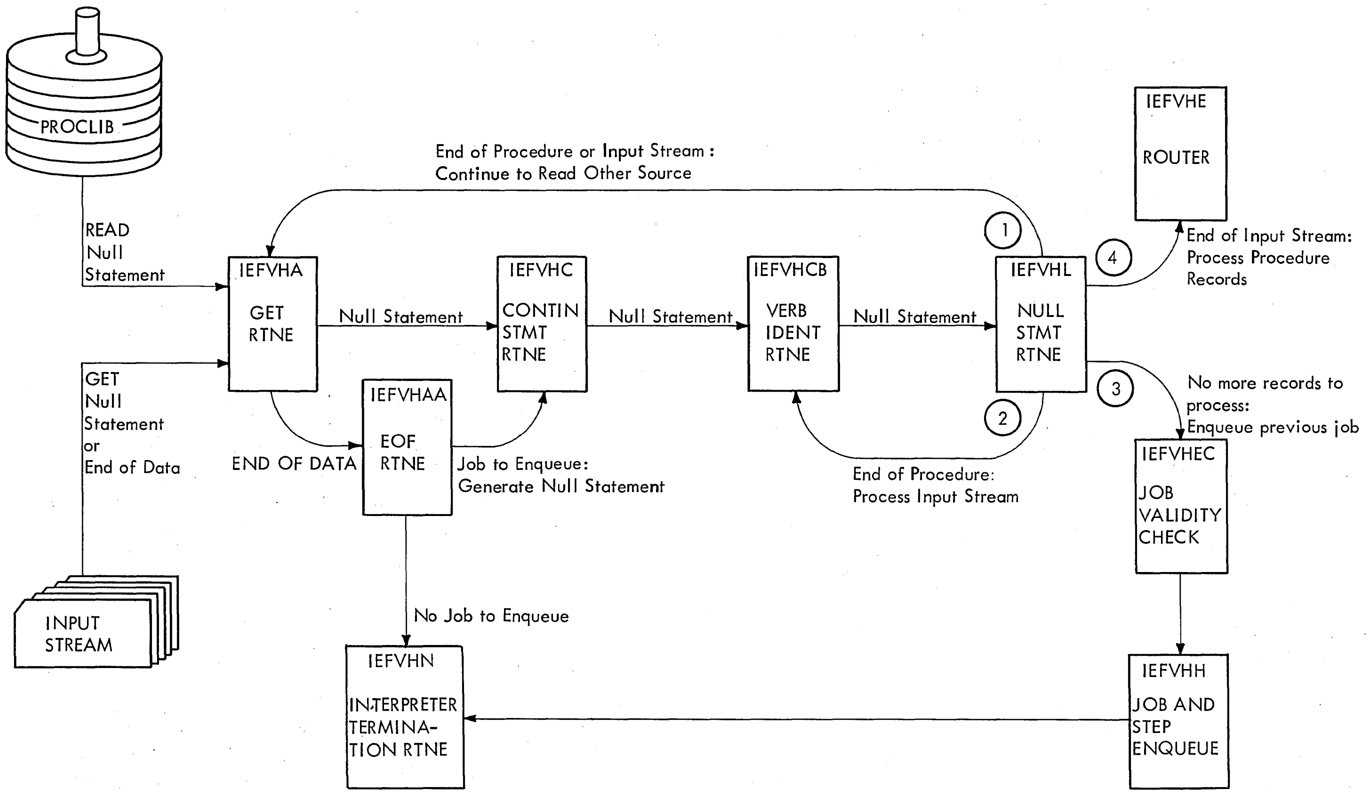

PROCLIB
READ Null Statement
GET Null Statement or End of Data
INPUT STREAM
IEFVHA GET RTNE
END OF DATA
IEFVHAA EOF RTNE
Job to Enqueue: Generate Null Statement
No Job to Enqueue
IEFVHN INTERPRETER TERMINA-TION RTNE
Null Statement
IEFVHC CONTIN STMT RTNE
Null Statement
IEFVHCB VERB IDENT RTNE
Null Statement
IEFVHL NULL STMT RTNE
End of Procedure or Input Stream : Continue to Read Other Source
1
2
End of Procedure: Process Input Stream
4
End of Input Stream: Process Procedure Records
IEFVHE ROUTER
3
No more records to process: Enqueue previous job
IEFVHEC JOB VALIDITY CHECK
IEFVHH JOB AND STEP ENQUEUE

Figure 66. End-of-Data and Null Statement Processing

② If the null statement represents the end of a procedure, but there are additional input stream records to process, control is passed to the Verb Identification routine to process the current record from the input stream.

③ If there are no more records to be processed in either the input stream or the procedure, control is passed to the Job Validity Check routine, so that the last job can be enqueued.

④ If there are no more input stream records to process, but there are additional records in the procedure, control is passed to the Router routine.

When the last job has been enqueued, control is passed to the Interpreter Termination routine.

Processing Control Statements

As shown in Figure 68, when a record containing the characters "//" in the first two positions is read, control is passed to the Continuation Check routine (module IEFVHC). If the current record begins in positions 4 through 16, it is expected to be a continuation of the preceding statement. The Continuation routine inspects the current record to determine whether it is a continuation. If so, control is passed to the Pre-scan Preparation routine; if not, or if no continuation was expected, control is passed to the Verb Identification routine.

The Verb Identification routine (module IEFVHCB) identifies the type of control statement that has been encountered, and processes them as follows:

- If the statement is an EXEC PROC statement, switches are set that cause the Interpreter Get routine to read a statement from the procedure library or from the in-stream procedure, then control is passed to the Router routine.
- If the statement is a PROC statement, and if it is the first statement in an in-stream procedure, control is passed to the In-stream Procedure Router routine (module IEFVINA) to process the procedure. Then, or if the PROC statement is the first statement in a cataloged procedure, control is passed to the Router routine. If the PROC statement is not the first statement in a procedure, the routine sets the job-failed bit in the JCT, and the statement is not processed further; control is passed to the Interpreter Get routine (module IEFVHA).
- If the statement is a JOB, EXEC, or DD statement, control is passed to the Router routine (module IEFVHE).
- If the statement is a PEND statement, control is passed to the Interpreter Message routine (module IEFVGM), which issues an error message.
- If the statement is a null statement, control is passed to the Null Statement routine (module IEFVHL).
- If the statement appears to have a valid format, yet does not have one of the valid JCL statement operators (JOB, EXEC, PROC, PEND, and DD), and is not a null statement, control is passed to the Command routine.

The Command routine (module IEFVHM) verifies that the verb is one of those allowed in the input stream. It also determines the disposition and continues as follows:

If disposition is:	The Command routine:
0	Causes the command to be executed.
1	Displays the command (via a WTO to the master console operator, and causes the command to be executed.
2	Displays the command (via a WTO) to the master console operator, requests his authorization to execute the command (via a WTOR), and if the reply is positive, causes the command to be executed.
3	Ignores the command, returning control to the Interpreter Get routine (module IEFVHA).

To execute the command, the Command routine places the command authority of the reader into register 0, makes the register negative to indicate that the command came from the job stream, and issues an SVC 34 to execute the Command Scheduling routine. The Command routine then passes control to the Interpreter Get routine (module IEFVHA.)

Processing In-Stream Procedures

When the Verb Identification routine encounters a PROC statement that is the first statement in an in-stream procedure, it obtains storage for a QMPA and for the in-stream procedure work area, then passes control to the In-stream Procedure Router routine (module IEFVINA) to begin processing the in-stream procedure. When it is first entered, the In-stream Procedure Router routine places EODAD and SYNAD exits in the QSAM DCB. Then the routine passes control to the In-stream Procedure Syntax Check routine (module IEFVINE).

The Syntax Check routine checks the validity of the label and operation fields in the PROC statement and passes a return code to the In-stream Procedure Router routine. Depending on the code, the Router routine continues processing as follows:

- If the return code is 0, 12, or 16, the PROC statement contains syntax errors: a zero indicates that the procedure name has been omitted from the PROC statement; a 12 indicates an invalid label; and a 16 indicates an error in the placement of comments. The routine sets the job-failed bit in the JCT and uses the Interpreter Message module (module IEFVGM) to issue the appropriate error message. Then it passes control to the Interpreter Get routine (module IEFVHA) to read the next statement. When the PEND statement ending the in-stream procedure is encountered, control passes to the Interpreter Message routine (module IEFVGM) which issues an error message indicating that the PEND statement is out of sequence.

- If the return code is 8, there are no syntax errors in the PROC statement. The routine initializes the QMPA and uses the Queue Management Assign/Start routine (module IEFQAGST) to prepare the QMPA for processing. Then the routine checks the IWA to determine if the procedure being processed is the first in-stream procedure. If it is, control passes to the In-stream Procedure Directory Build routine (module IEFVINC). If it is not, the routine builds a parameter list for the procedure (see Figure 67) and passes control to the In-stream Procedure Directory Search routine (module IEFVINB).

The Directory Search routine scans the entries in the in-stream procedure directory (located in the in-stream procedure work area) searching for the procedure name specified in the PROC statement. If the procedure name is found, the routine obtains from the directory the TTR of the first record containing the procedure and places the TTR in the return code field of the parameter list. If the procedure is not found in the directory, the routine sets a return code of zero in the parameter list. Control then passes to the In-stream Procedure Directory Build routine.

The In-stream Procedure Directory Build routine enters the procedure name in the directory. It also uses the Queue Management Interface routine (module IEFVHQ) and the Queue Management Assign routine (module IEFQASGQ) to assign a record. The record address returned from the Assign routine is placed in the directory following the procedure name. If the directory already contains fifteen entries, control returns to the In-stream Procedure Router routine, which creates an SMB for the error message that will be issued by the Interpreter Message module and sets the job-failed bit in the JCT. The routine finally passes control to the Interpreter Get routine to read the next statement.

Address of Parameter List	4
Procedure Name	8
Address of In-Stream Procedure Directory	4
Address of IWA	4
Return Code	4

Figure 67. In-Stream Procedure Parameter List

After the Directory Build routine enters the procedure entry in the directory, the In-stream Procedure Router routine creates an SMB for the statement (if MSGLEVEL = 1 or 2 is specified) and passes control to the Record Compression routine (module IEZNCODE) to compress the statement. If the record is too large for the compression work area (located in the in-stream procedure workarea), the In-stream Procedure Router routine uses the Queue Management Assign routine (module IEFQASGQ) to assign the record to a queue entry. (The address of the record is placed in the in-stream procedure work area following the procedure name to point to the next record of the procedure.) Then the record is written to the job queue by the Queue Management Read/Write routine (module IEFQMRAW), and the Record Compression routine receives control again to compress the record. After the record has been compressed, the In-stream Procedure Router routine reads the next statement.

First the routine checks the statement to determine if the characters "//" are in the first two positions. If not, the statement is considered to be a data statement, and the routine generates a SYSIN DD * statement, sets the job-failed bit in the JCT and the job flush bits in the IWA, and returns control to the Interpreter Get routine to flush the data. If the "//" characters exist, the routine scans the verb field to identify the type of control statement and processes them as follows:

- If the statement is either a JOB or a NULL statement, the routine releases the storage occupied by the parameter list, reinitializes the pointers to the statement being processed and its control information, and passes control to the Verb Identification routine (module IEFVHCB). If it is a NULL statement and MSGLEVEL = 1 or 2 is specified, the Router routine creates an SMB for the statement before it passes control to the Verb Identification routine.

- If the statement is a PEND statement, the routine passes control to IEFVINE to check the statement for syntax errors. If an error exists, an error message is issued. Then, or if no error exists, the routine creates an SMB (if MSGLEVEL = 1 or 2 is specified), frees the storage used by the parameter list, and returns control to the Interpreter Get routine to read the next statement.

- If the statement is a DD statement, the routine checks the operand to determine whether it is a DD * or DD DATA statement. If it is either of these, the routine sets the data flush bit in the IWA and the job-failed bit in the JCT, and returns control to the Interpreter Get routine to flush the data. Otherwise, the routine creates an SMB (if MSGLEVEL = 1 or 2) and passes control to the Record Compression routine to compress the record. Then another statement is read and the processing continues until a JOB, DD*, DD DATA, data, PEND, or NULL statement is encountered. After all statements have been compressed, control returns to the Verb Identification routine.

When the Verb Identification routine encounters an EXEC PROC statement, the EXEC Statement Processor (module IEFVEA) receives control to process it. The routine searches the in-stream procedure directory in the in-stream procedure work area to determine if the specified procedure is an in-stream procedure. If no entry is found in the directory, the routine omits in-stream procedure processing.

If an entry exists in the directory, the routine saves the address of the BPAM access method in the IWA and places the address of the In-stream Procedure Decompression Interface routine (module IEFVIND) in the BPAM DCB. The Decompression Interface routine receives control when the next statement is read. If it is the first statement of the procedure, the routine reads the first procedure record from the job queue into the work area, builds a parameter list and passes control to the Record Decompression routine (module IEZDCODE) to decompress the record. With each READ following the first, the Record Decompression routine receives control and decompresses the record. This continues until all records have been decompressed. Then the routine checks the address of the next record of the compressed procedure in the in-stream procedure work area to determine whether the in-stream procedure is contained in another job queue record. If the field is not zero, another record exists; the routine reads in the other record and decompresses it. If the field contains zero, the procedure is finished; therefore, the routine returns the address of the BPAM access method to the BPAM DCB and returns control to the Interpreter Get routine.

Processing JOB, EXEC, PROC, and DD Statements

When the Verb Identification routine determines that the statement is a JOB, EXEC or DD statement, it passes control to the Router routine (module IEFVHE -- see Figure 68). The Router routine determines whether there are tables from a previous step to be placed in the queue before the current statement can be processed.

If the router is entered with an EXEC statement in the buffer, the tables describing the previous step must be placed in the job's queue entry; control is passed to the Job and Step Enqueue routine (module IEFVHH).

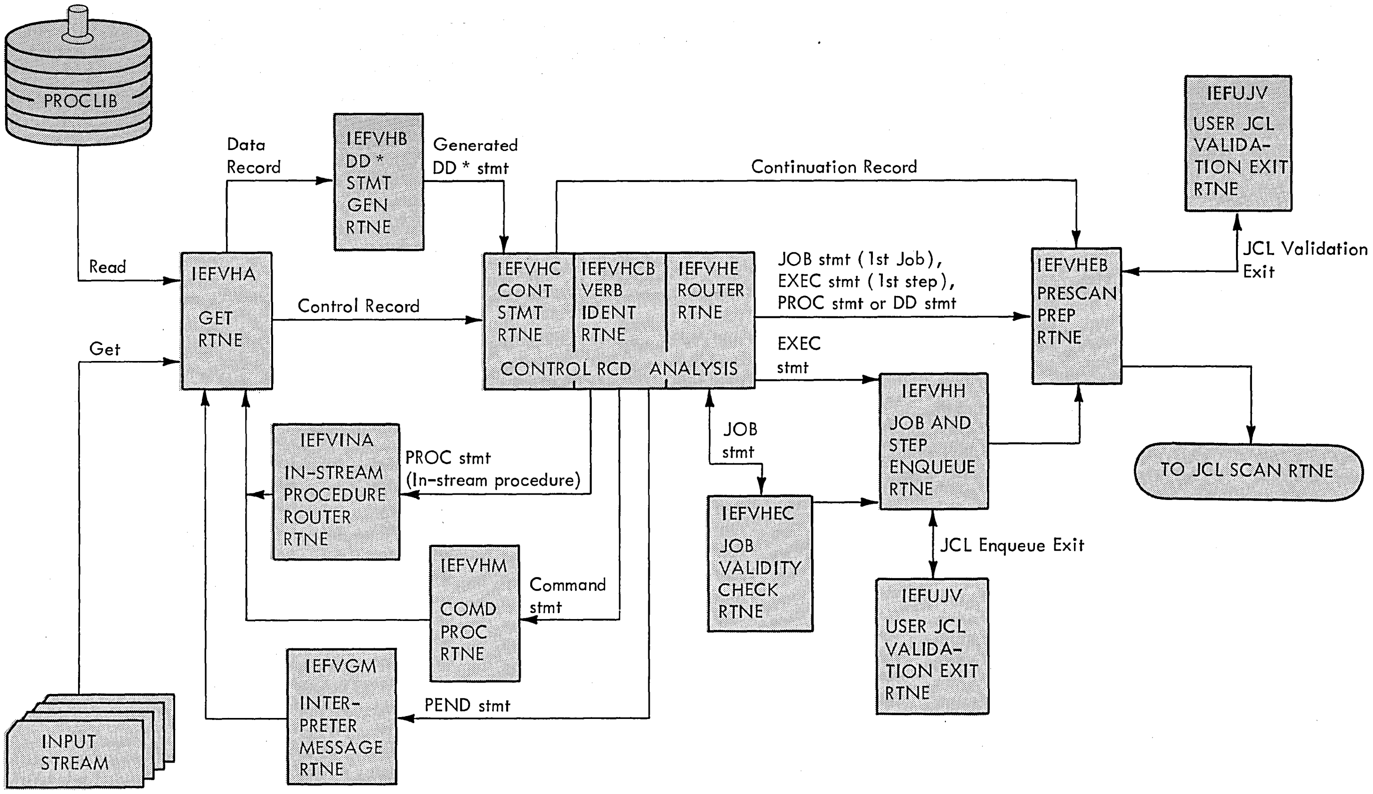

PROCLIB
Data Record
IEFVHB DD * STMT GEN RTNE
Generated DD * stmt
Continuation Record
IEFUJV USER JCL VALIDA-TION EXIT RTNE
Read
IEFVHA GET RTNE
Control Record
IEFVHC CONT STMT RTNE
IEFVHCB VERB IDENT RTNE
IEFVHE ROUTER RTNE
CONTROL RCD ANALYSIS
JOB stmt (1st Job), EXEC stmt (1st step), PROC stmt or DD stmt
IEFVHEB PRESCAN PREP RTNE
JCL Validation Exit
Get
EXEC stmt
IEFVHH JOB AND STEP ENQUEUE RTNE
TO JCL SCAN RTNE
IEFVINA IN-STREAM PROCEDURE ROUTER RTNE
PROC stmt (In-stream procedure)
JOB stmt
IEFVHEC JOB VALIDITY CHECK RTNE
JCL Enqueue Exit
IEFVHM COMD PROC RTNE
Command stmt
IEFUJV USER JCL VALIDA-TION EXIT RTNE
IEFVGM INTER-PRETER MESSAGE RTNE
PEND stmt
INPUT STREAM

Figure 68. Processing JOB, EXEC, PROC, PEND, DD, and Command Statements

If the statement in the buffer is a JOB statement, the previous step is the last step of a job; control is therefore passed to the Job Validity Check routine (module IEFVHEC).

If the statement in the buffer is a PROC or DD statement, or if it is an EXEC statement representing the first step in a job, or if it is a JOB statement representing the first job in the input stream, there are no tables to be written into the queue, and control is passed to the Pre-scan Preparation routine.

The Pre-scan Preparation routine (module IEFVHEB) is entered when the statement to be processed is a JOB, EXEC, PROC or DD statement.

If the statement is a JOB statement, the Pre-scan Preparation routine passes control to the Queue Manager Interface routine (module IEFVHQ), which uses the Queue Management Assign and Start routines to start an input queue entry with an assignment of five records (six, if SMF is included in the system).

On the return, the Pre-scan Preparation routine (module IEFVHEB) stores the address of the record assigned to the job management record (JMR) in the JMR pointer field of the JCT and issues the TIME BIN macro instruction to obtain the reader start time and date. The routine initializes the JMR, then passes control to the user's JCL Validation routine (if one has been supplied) or to the IBM-supplied dummy routine.

On the return from the JCL Validation Exit routine (module IEFUJV), the Pre-scan Preparation routine inspects the return code and if it indicates that the job is to be canceled, sets the job-failed bit on in the JCT. The routine then stores the queue addresses assigned to the JCT and SCT, then uses the Message Writing routine (module IEFVGM) to copy the JOB statement into an SMB, where it will be made available to the programmer via a system output writer.

If the statement is a PROC statement, the Pre-scan Preparation routine converts it to an EXEC PROC statement.

If the statement is not a JOB statement, the Pre-scan Preparation routine examines the JCL message level in the JCT. If the JCL message level of the job is one (MSGLEVEL = (1,x)), the Message Writing routine places the other JCL statements and the cataloged procedure statements for the job into SMBs. Futhermore, if any of the JCL statements contain symbolic parameters, the substituted JCL statement is entered in an SMB following the original input JCL statement. If the JCL message level of the job is two, only input JCL statements are entered into SMBs. Next, the routine passes control to the user's JCL Validation routine (if one has been supplied), and on the return it inspects the return code. If the return code specifies that the job is to be canceled, the routine sets the job-failed bit in the JCT.

When it has completed statement processing, the Pre-scan Preparation routine finally passes control to the JCL Scan routine (module IEFVFA), which converts statements to internal text, and passes them to the appropriate processor, so that the tables can be constructed.

Queue Entry Processing

When the presence of a JOB or null statement in the input stream indicates that the input queue entry describing the previous job is to be enqueued, the Job Validity Check routine (module IEFVHEC) is entered. The routine determines whether the job to be enqueued has any steps; if so, control is passed to the Job and Step Enqueue routine. If not, the Validity Check routine constructs a dummy SCT and sets the job-failed bit on before passing control to the Job and Step Enqueue routine.

The Job and Step Enqueue routine (module IEFVHH) is entered from the router when the presence of an EXEC statement indicates that the tables representing the previous step are to be placed in a queue, and from the Job Validity Check routine when the presence of a JOB or null statement indicates that the step was the last step of a job. If the statement is not in a procedure, the Job and Step Enqueue routine scans the symbolic parameter table (SYMBUF) to determine whether all symbolic parameters that have been assigned a value have also been defined. If not, the routine sets the job-failed bit in the JCT.

The Job and Step Enqueue routine inspects switches in the IWA to determine which tables are to be placed in the queue, then passes control to the Queue Management Interface routine (module IEFVHQ) to have each table written to the queue by queue management.

If the step whose tables are to be placed in the queue is the last step in a job, a switch in the IWA indicates that the JMR and JCT are to be written. When the tables describing the step have been placed in the queue, the Job and Step Enqueue routine determines whether the interpreter is to enqueue jobs. If not, the routine instructs the Queue Management Interface routine to have the JMR and JCT written by the queue management Read/Write routine, and on the return it passes control to the interpreter Termination routine.

If the interpreter is to enqueue jobs, the Job and Step Enqueue routine passes control to the user's JCL Validation routine (module IEFUJV), then issues a TIME BIN macro instruction to obtain reader stop time. The routine stores the time and date in the JMR, then instructs the Queue Management Interface routine to have the JMR written in the queue data set. On the return, it instructs the Queue Management Interface routine to have the JCT written by the Queue Management Enqueue routine. This causes the JCT to be placed in the entry, thus completing it, and it causes the entry to be enqueued, thus making it available for selection by an initiator. If the job-failed bit in the JCT is on, the routine enqueues the entry in the hold queue at a high priority and issues a CANCEL command for the job.

After the Job and Step Enqueue routine places the JCT in the queue, it determines whether in-stream procedures were included in the job. If they were, the routine releases all storage used by the procedures, deletes job queue records used for the in-stream procedures, and reinitializes pointers to the storage areas in the IWA. The Job and Step Enqueue routine passes control to the Job and Step Enqueue Routing routine (module IEFVKG). This routine determines from the CSCB whether a STOP RDR command has been issued. If so, or if there are no more records in the input stream, control is passed to the Interpreter Termination routine. Otherwise, control is passed to the Job and Step Enqueue Housekeeping routine.

The Job and Step Enqueue Housekeeping routine (module IEFVHHB) reinitializes the JCT area of the IWA. If there is a procedure to be processed, it then passes control to the Pre-scan Preparation routine which processes the statement currently in the buffer. If there is no procedure to be processed, control is passed to the Get routine (module IEFVHA).

Post-Processing Entry

The Control routine is reentered at the Post-scan routine (module IEFVHF) from the JCL Scan routine if: a continuation statement is expected, if the statement scanned was an overriding statement, or if a JCL error was detected on a DD statement. It is entered from a Statement Processor routine when the processing of a statement is complete or a JCL error was detected on a JOB or EXEC statement. The Post-scan routine determines the conditions under which it was entered, then passes control (as shown in Figure 69) to the appropriate Control routine module:

- If a continuation statement is expected, control is passed to the Interpreter Get routine to read the statement.
- If a continuation statement was expected but not received, control is passed to the Verb Indentification routine to process the statement actually received (module IEFVHCB).
- If an overriding statement has been processed by the JCL Scan routine, the overridden statement must be scanned before the Statement Processor routine is entered. The overridden statement is in the buffer, and control is passed to the Pre-scan Preparation routine (module IEFVHEB).
- If a JCL error was encountered, the job-failed bit has been set on. The remaining statements in the job (except for procedure library statements) will be processed by the interpreter, so that any other errors may be found, but the job will not be run. Control is passed to the Interpreter Get routine, and processing continues.
- If the statement has been successfully processed, control is passed to the Interpreter Get routine.
- If the statement processed was a DD* or DD DATA statement, control is passed to the Interpreter Spool routine.

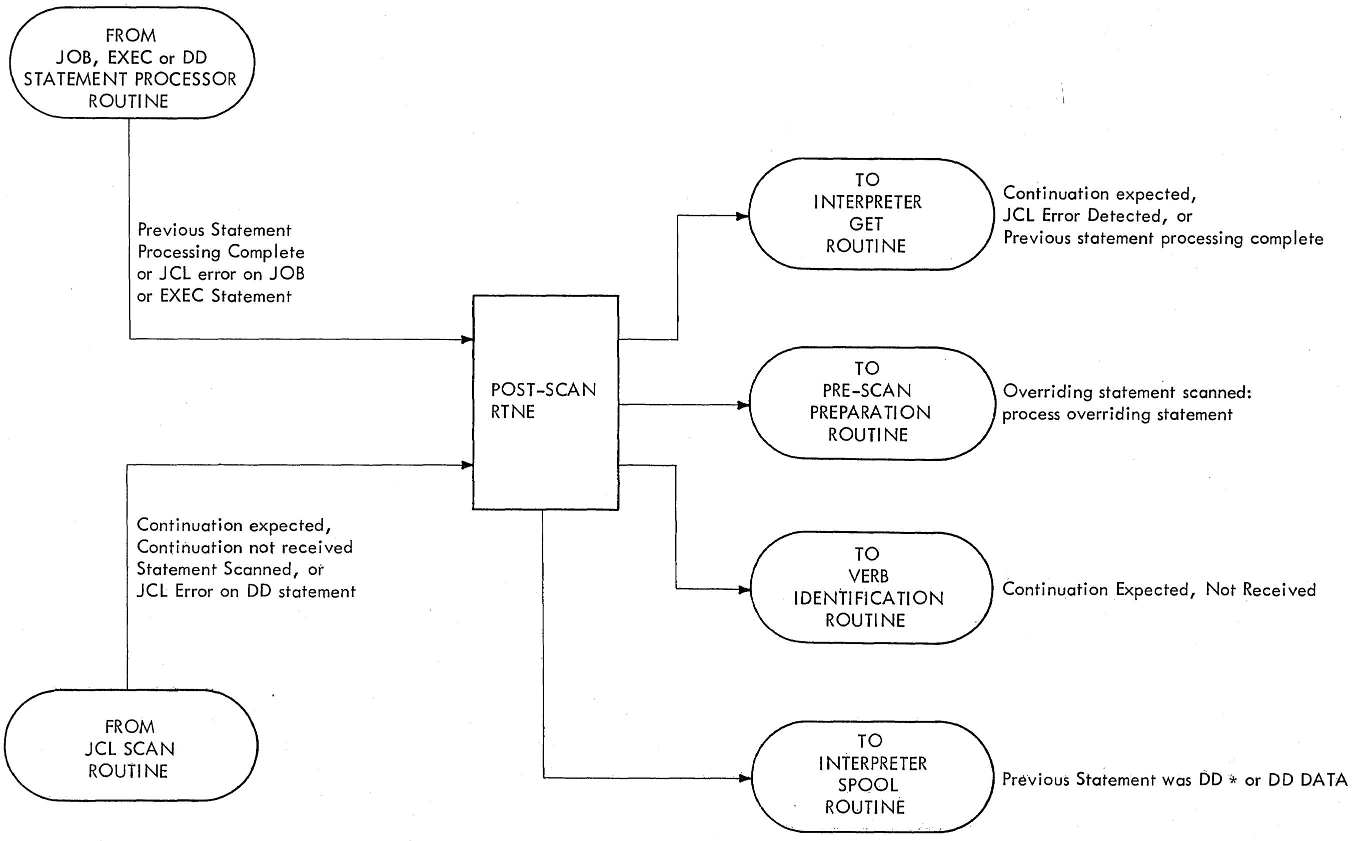

FROM JOB, EXEC or DD STATEMENT PROCESSOR ROUTINE
Previous Statement Processing Complete or JCL error on JOB or EXEC Statement
POST-SCAN RTNE
Continuation expected, Continuation not received Statement Scanned, or JCL Error on DD statement
FROM JCL SCAN ROUTINE
TO INTERPRETER GET ROUTINE
Continuation expected, JCL Error Detected, or Previous statement processing complete
TO PRE-SCAN PREPARATION ROUTINE
Overriding statement scanned: process overriding statement
TO VERB IDENTIFICATION ROUTINE
Continuation Expected, Not Received
TO INTERPRETER SPOOL ROUTINE
Previous Statement was DD * or DD DATA

Figure 69. Post-Processing Entry

Processing System Input Data

The Interpreter Spool routine (module IEFVHG) is entered from the post-scan routine when a DD* or DD DATA statement has been processed. Using a JFCB created in module IEFVDA, the DD Statement Processor routine, (which also uses DADSM to allocate space for the SYSIN data set) the routine builds and opens an output DCB. It then uses the GET macro instruction to read records from the input stream, and determines whether the first two characters of each record are '//'. If the first two characters are '//', and module IEFVGH was entered after a DD DATA statement was processed, it uses the PUT macro instruction to write the record to the output data set. If the first two characters are '//' and the module was entered after a DD* statement was processed, it passes control to the Interpreter Get routine (module IEFVHA).

As the Interpreter Spool routine reads each record whose first two characters are not '//' from the input stream, it determines whether the DLM parameter has been specified. If it has, the routine checks the first two characters for the DLM value. If the DLM value has not been specified, and if the characters are not '/*', the routine uses the PUT macro instruction to write the record to the output data set. If the routine finds the DLM value, or if it finds that the DLM value has not been specified but the characters '/*' are present, it passes control to the Interpreter Get routine.

If an end of data condition is encountered, control is passed to the EODAD Exit routine. If SMF is included, the Interpreter Spool routine places the record count of the SYSIN data set in the step counting field of the SCT at the end of SYSIN data.

I/O Error Processing

When an I/O error is encountered during interpreter processing, the job-failed bit in the JCT, and the I/O error and job-flush bits in the IWA are set on, and control is passed to the Message Writing routine (module IEFVHR). This routine uses the WTO macro instruction to inform the operator, then places an error message in the current SMB. Control is returned to the Interpreter Get routine, which flushes (reads, but does not process) the remaining input stream records for the current job, then begins normal processing with the next JOB statement.

SCANNING THE JCL STATEMENT

The Job Control Language Scanning routine (module IEFVFA) converts a JCL record into a coded internal list (see Figure 70). When it has accumulated a complete JCL statement (including continuations and overrides), it then passes the list to a Statement Processing routine. An example showing the scanning and encoding of a DD statement follows this section.

Each statement is scanned from left to right. The Scan routine is able to identify the existence of a name field, keywords and symbolic parameters, and one level of subparameters following a keyword.

As the statement is examined, the name field, keywords, and symbolic parameters are identified and the keywords are looked up in the scan dictionary (see Figure 71). For each valid keyword the scan dictionary entry contains the corresponding one-byte binary "key", and lists the keys of any mutually exclusive keywords (the DDNAME and SPACE parameters, for example, are mutually exclusive). The entry also lists the keys of any minor keywords associated with the keyword that the entry represents; SEP, for example, is a minor keyword of the UNIT parameter and is listed as a minor keyword in the UNIT entry of the scan dictionary. The list of mutually exclusive keys is used for error checking and the minor keys are overridden when the corresponding major key is overridden.

Key	Number	Length	Positional Parameter	Count	Length	Sub-Parameter

Figure 70. Internal List Entry Format

Notes to Figure 70

Key is the one byte binary code that represents a keyword.

Number is a one byte binary number that specifies the number of positional parameters in the entry. Its high-order bit is always off.

Length is a one byte number that specifies the length of the parameter and subparameters that follow it. Its high-order bit is always off.

Count is a one byte binary number that specifies the number of subparameters in the entry. Its high-order bit is always on.

Positional Parameter contains the positional parameter.

Note: The format of a list entry is variable, depending on the presence and number of positional parameters and subparameters.

When the correct scan dictionary entry has been found, the Scan routine determines whether the keyword has been encountered previously, or whether a mutually exclusive keyword has been encountered, by testing the appropriate bits in the duplicate table.

The duplicate table is a 32-byte table that contains a bit for each key. The position of the bit in the table corresponds to the key.

Length of Entry	Keyword	Key	Mutually Exclusive Key	Overridden Key

Scan Dictionary Entry Format

Figure 71. Scan Dictionary Entry Format

Notes to Figure 71

Length of Entry is a one byte binary number that specifies the length, in bytes, of the scan dictionary entry (including the length of entry field).

Keyword contains the keyword specified in this entry.

Key is a one byte binary code that represents the keyword specified in this entry.

Mutually Exclusive Key contains the key that represents a keyword that may not be used in a statement that contains a keyword specified in this entry. The high-order bit in this field is always off for DD statement keywords.

Overridden Key contains the key that represents a minor keyword of the keyword specified in this entry. The high order bit in this field is always on.

Note: The format of a scan dictionary entry is variable, depending on presence and number of mutually exclusive and overridden keys.

When it makes an entry in the internal list, the Scan routine turns on the bit that corresponds to the key it is processing. It also turns on the bits that correspond to any mutually exclusive keys, as defined in the scan dictionary entry. Thus, if a bit in the table is on, it means that the key, or a mutually exclusive key, has been encountered previously.
If this condition occurs during a procedure

merge, the field being processed has been overridden and the Scan routine proceeds to the next field. In all other cases, the Scan routine turns on the job-failed bit in the JCT, and passes control to the Post-scan routine (module IEFVHF).

Processing Symbolic Parameters

A programmer defines symbolic parameters, either in statements that override procedure statements or in the statements of the procedure. He must also assign values to symbolic parameters by coding them in an EXEC statement that calls a procedure or in a PROC statement.

The Scan routine may therefore encounter symbolic parameters in EXEC statements that call procedures, in input stream statements that override procedure statements, or in statements from a procedure. (Symbolic parameters may also be found in PROC statements. Such statements, however, are modified in the Pre-scan Preparation routine (module IEFVHEB) of the Interpreter Control routine, and appear to the Scan routine as EXEC statements that call a procedure. The EXEC Statement Processor routine, however, tests a bit set by the Verb Identification routine, determines whether the statement is a PROC statement, and if so, ignores it.)

If the statement is an execute procedure statement, the Scan routine uses the Symbolic Parameter Processing routine to make an entry in a table of symbolic parameters and their assigned values; otherwise, the Scan routine uses the Symbolic Parameter Processing routine to extract a value from the table, and to substitute the value for the symbolic parameter.

When a symbolic parameter appears in an execute procedure statement, it has the format of an EXEC statement keyword. The Scan routine searched the scan dictionary, and if no matching EXEC statement keyword is found, it assumes that the keyword and its associated positional parameter are a symbolic parameter and its value assignment, respectively. A BALR instruction is executed to pass control to the Symbolic Parameter Processing routine, to create an entry in the symbolic parameter table.

When it is entered, the Symbolic Parameter Processing routine (module IEFVFB) verifies that the EXEC statement calls a procedure, then determines whether a symbolic parameter table, or SYMBUF (see Figure 72) has been initialized since the current procedure was called.

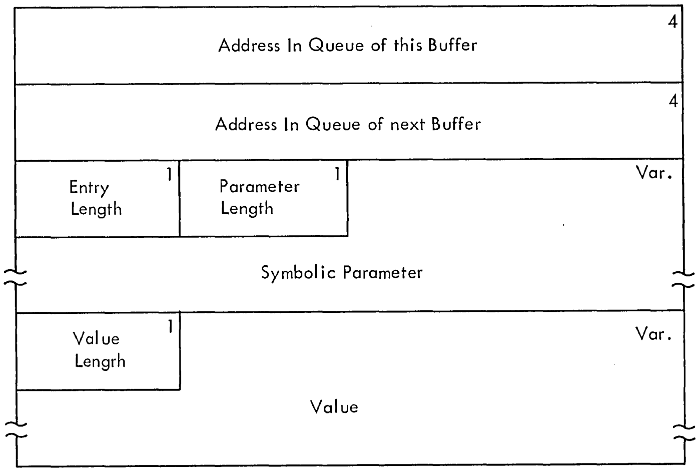


Figure 72. Symbolic Parameter Table Buffer

Note: The high-order bit of the parameter length field is set on when the value assignment is made, and set off when the parameter is defined in a statement.

If no SYMBUF has been initialized, the routine initializes one (in its local work area), and creates an entry containing the symbolic parameter and its assigned value.

If a SYMBUF has been initialized, the Symbolic Parameter Processing routine searches for an entry corresponding to the current symbolic parameter. If no such entry exists, the routine creates one; if the entry does exist, the routine ignores the current value assignment.

When the value assignment processing is complete, the Symbolic Parameter Processing routine returns control to the Scan routine, which continues the scan of the statement.

When a symbolic parameter appears in an input stream statement that overrides a procedure statement, or in a statement in a procedure, the parameter is immediately preceded by a single ampersand (&). The scan routine branches to the Symbolic Parameter Processing routine, which substitutes the corresponding value from the SYMBUF for the parameter.

The Symbolic Parameter Processing routine (module IEFVFB) uses the character string following the ampersand as a search argument to search the SYMBUF. If it finds a matching entry, the routine extracts the value assigned to the parameter, and stores it in an intermediate text buffer. If no matching entry is found, the routine stores the ampersand and the character string in the intermediate text buffer.

In either case, the routine moves any concatenated text, as well as the next major delimiter (comma or blank) following the symbolic parameter, into the intermediate text buffer. It then sets pointers to cause the Scan routine to process in the buffer.

The routine then examines the JCL message level in the JCT. If the JCL message level of the job is one (MSGLEVEL = (1,x)), the JCL statement as it appears after substitution of symbolic parameters is entered in an SMB following the original input JCL statement. Control then returns to the Scan routine.

The Scan routine, (module IEFVFA) scans the text in the intermediate text buffer as if it were in the original statement, and makes the appropriate entries in the internal list. When it has completed processing in the intermediate text buffer, the Scan routine continues the scan of the original statement.

When the scan is complete, control is passed to the appropriate JCL Statement Processor routine.

Example:

The JCL Scan routine encounters the following source statement.

```
//SYSUT1 DD DSNAME=LINKEDIT.WORK,UNIT=190,                              C
//             SPACE=(TRK,(30,10)),VOLUME=SER=111111
```

1. The name field (SYSUT1) is identified as such because of its position. It is considered a parameter of the DD keyword, and is encoded as follows:

Key	Number	Length	Parameter
6E	01	06	E2 E8 E2 E4 E3 F1

2. The DSNAME= field is found in the scan dictionary entry shown below:

Length	Keyword	Key	Mutually Exclusive keys	
0B	C4 E2 D5 C1 D4 C5 7E	4A	49	4B

3. It is encoded and placed in the list as shown below:

Key	Number	Length	Parameter
4A	01	0D	D3 C9 D5 D2 C5 C4 C9 E3 4B E6 D6 D9 D2

4. The UNIT= field is found in the scan dictionary entry shown below:

Length	Keyword	Key	Mutually Exclusive Key	Overridden Key	Overridden Key
0A	E4 D5 C9 E3 7E	41	49	CD	CE

5. It is encoded and placed in the list as shown below:

Key	Number	Length	Parameter
41	01	03	F1 F9 F0

6. The SPACE= field is found in the dictionary entry shown below:

Length	Keyword	Key
08	E2 D7 C1 C3 C5 7E	47

7. It is encoded and placed in the list as shown below:

Key	Number	Length	Parameter	Count	Length	Parameter	Length	Parameter
47	02	03	E3 D9 D2	82	02	F3 F0	02	F1 F0

8. The VOLUME= field is found in the dictionary entry shown below:

Length	Keyword	Key	Mutually Exclusive Key	Overridden Key	Overridden Key
0C	E5 D6 D3 E4 D4 C5 7E	43	49	CF	CA

9. It is encoded and placed in the list as shown below:

Key	Number
43	00

Since there are no positional parameters associated with the VOLUME keyword, the number field is 00, and it terminates the entry.

10. The serial number field (SER=) is found in the scan dictionary entry shown below:

Length	Keyword	Key	Mutually Exclusive Key
07	E2 C5 D9 7E	4F	50

11. It is encoded and placed in the list as shown below:

Key	Number	Length	Parameter
4F	01	06	F1 F1 F1 F1 F1 F1

12. Since the serial number field is the last field in the statement, the list is closed with the entry:

Key
FE

The Scan routine then passes control to the DD Statement Processor routine (IEFVDA).

PROCESSING JCL STATEMENTS

When a statement has been scanned, and its contents placed in an internal text buffer, tables must be built from the internal text. This function is performed by the JOB Statement Processor routine (module IEFVJA), the EXEC Statement Processor routine (module IEFVEA), and the DD Statement Processor routine (module IEFVDA). These three routines are similar in construction (see Chart 06); each processor consists of a single control section containing a Header routine, a Keyword routine for each keyword in the statement, and a Cleanup routine.

When a Statement Processor routine is first entered, the Header routine performs initializing functions, which include clearing the storage area occupied by the tables to be created by the routine (except for fields filled in by previously executed routines), and initializing the local work area (LWA). It then uses a BALR instruction to pass control to the Get Parameter routine, which finds the first parameter, performs basic error checking, then passes control to the appropriate Keyword routine.

Each Keyword routine controls the processing of the positional parameters and subparameters associated with a given keyword. The routine is entered initially when the Get Parameter routine encounters its keyword, and again as each positional parameter and subparameter is found. In some cases, the required processing is done directly by the Keyword routine; in most cases, however, the keyword routine passes control to the Test and Store routine, which processes the parameter in accordance with the description in the parameter descriptor table (PDT) and returns control to the Keyword routine. Control is then passed to the Get Parameter routine for the next parameter.

When the last parameter in the statement has been processed, or when the Test and Store routine or Get Parameter routine finds an error, control is passed to the cleanup portion of the JCL statement processor.

Each Cleanup routine uses the Message routine to write any error messages to the programmer. In addition, the Cleanup routines perform the processing described below:

- The JOB Statement Processor Cleanup routine checks for the presence of programmer name and account number, and uses the Queue Manager Interface routine to write out the job account control table (ACT).
- The EXEC Statement Processor Cleanup routine determines whether the region size parameter has been placed in the SCT. If not, it extracts the region size specification from the JCT (if there is one), or stores the minimum region size specified when the system was generated in the SCT. If the EXEC statement specifies "PROC=", the routine uses the Queue Management Interface routine to write out the last override table; if the statement was in a procedure, the routine reads the appropriate override table into main storage, and stores overriding information in the SCT.
- The DD Statement Processor Cleanup routine performs cleanup functions for all DD Statements, and in addition it passes control to the SYSIN Processor and Get Parameter routines to process DD * and DD DATA statements.

The SYSIN Processor routine (module IEFVSD12) creates a JFCB for the input stream data set using the IEFDATA DD statement in the reader procedure as a pattern. If the DD * contains overiding DCB parameters, the routine processes them and allocates direct access space for the data set. The routine creates a SER= entry in the internal text buffer, using the serials of the volumes on which space was allocated, then returns control to the DD Statement Processor Cleanup routine.

The Cleanup routine passes control to the Get Parameter routine, which recognizes the SER= entry in the internal text buffer, and passes the parameters to the Keyword routine. The Keyword routine processes the volume serials, then returns control to the DD Statement Processor Cleanup routine.

When it is thus entered for the second time, or when it is entered to process other DD statements, the DD Statement Processor Cleanup routine sets initializing values in the JFCB, where no value has previously been set. It marks the disposition fields for implied dispositions, and sets bits to indicate whether the data set is public, private, temporary, or shareable. If the DSNAME keyword was omitted, or if its parameter is "&", the routine generates a data set name. It uses the Queue Manager Interface routine (module IEFVHQ) to assign records in the queue for the SIOT and JFCB (unless the DDNAME or SYSOUT keyword was used in the statement), then writes the SIOT and JFCB into the assigned records. If the DDNAME keyword was used, the records have previously been assigned, and the JFCB and SIOT need only to be written out. If the SYSOUT keyword was used, the routine passes control to the SCD Construction routine (module IEFVSD13) routine, which assigns the space.

When a Cleanup routine has completed its processing, it passes control to the Interpreter Control routine, at the Post-scan routine (module IEFVHF).

AUXILIARY ROUTINES

During the performance of the reading task, the Interpreter routines must frequently perform functions common to several routines. These common functions are performed by a set of auxiliary routines, which are described below:

- The Get Parameter routine (module IEFVGK) is used by the Statement Processor routines. It searches for the next parameter in a statement, performs basic error checking, and passes control to the proper Keyword routine, with a pointer to the parameter.

- The Test and Store routine (module IEFVGT) is used by the Statement Processor routines. It processes the parameter as described in the parameter descriptor table (PDT) and passes control back to the Keyword routine.

- The Dictionary Entrance routine (module IEFVGI) is used by the Statement Processor routines. It makes entries for the dictionary used in refer-back processing.

- The Dictionary Search routine (module IEFVGS) is used by the Statement Processor routines. It searches the refer-back dictionary during refer-back processing.

- The Message routine (module IEFVGM) stores messages in system message blocks (SMBs) for transmittal to the programmer via system output writer.

- The Queue Manager Interface routine (module IEFVHQ) is used by those Interpreter routines that reserve space, write records to or read records from the queues.

- The Record Compression routine (module IEZNCODE) and the Record Decompression routine (module IEZDCODE) are used by the In-stream Procedure routines to compress and decompress JCL statements.

The Get Parameter Routine

The Get Parameter routine (module IEFVGK) is an auxiliary routine used by the JCL Statement Processor routines to find the next parameter in a statement, perform basic error checking of that parameter, and find and pass control to the appropriate Keyword routine with pointers to the parameter and to the appropriate parameter descriptor table (PDT) entry.

When the Get Parameter routine is initially entered, the only non zero portion of the auxiliary work area (AWA) is the address of the keyword branch table (KBT) and the address of the Processor Cleanup routine. The KBT (Figure 73) is a table of offsets that allows the Get Parameter routine to determine the actual main storage address of the appropriate Keyword routine and PDT entry. Additional fields in the table allow basic error checking to be done.

When the Get Parameter routine is entered to find the first parameter in a new statement, it extracts the base key (the key number that represents JOB, EXEC, or DD) from the text buffer (internal list) and stores it. There are three sets of key numbers: one set for the JOB statement, one set for the EXEC statement, and one set for the DD statement. The base key, which corresponds to the verb in the statement, is the highest number in the set. It is the offset of the last entry in the table from the first entry. Whenever the routine is entered, it subtracts the current key from the base key, multiplies the result by 6 (the size of a KBT entry), and adds the product to the machine address of the first entry in the table. The result is the machine address of the KBT entry corresponding to the current keyword.

<table>
<tr><td>Maximum Number of Parameters 1</td><td>Subparameter Check 1</td></tr>
<tr><td colspan="2">Offset to Keyword Routine 2</td></tr>
<tr><td colspan="2">Offset to PDT Entry 2</td></tr>
</table>

Figure 73. Keyword Branch Table Entry

The Get Parameter routine first finds the proper KBT entry, then determines whether the maximum number of parameters for the keyword has been exceeded, and stores the subparameter check byte in the AWA. Each bit in the subparameter check byte corresponds to a positional parameter; if the bit is on, it means that the corresponding parameter may have subparameters associated with it. For example, if the first positional parameter associated with a keyword were the only one that could consist of a subparameter list, the high-order bit in the field would be on. If the seventh and eighth positional parameters could have subparameters, the two low-order bits would be on.

The two offset fields are used to compute the actual main storage address of the appropriate Keyword routine and of the appropriate PDT entry; the positional parameter length, the parameter length byte address (in the internal text buffer) and the PDT entry address are placed in general registers, and control is passed to the Keyword routine.

On subsequent entries to the routine, the pointers are updated so that they point to the next operand (positional parameter or subparameter), and control is returned to the Keyword routine at the instruction after the branch to the Get Parameter routine. When the next keyword is encountered, however, the branch table is again used, and control is passed to a new Keyword routine.

The Test and Store Routine

The Test and Store routine (module IEFVGT) is an auxiliary routine used by the JCL Processor routines to determine the processing required for a parameter (as described in the PDT), and to perform that processing. When processing of a keyword is complete, control is returned to the appropriate Keyword routine.

The parameter descriptor table (Figure 74) included in each JCL statement processor describes the processing to be done for each parameter that may be found in the statement. There is an entry for each keyword, which begins with a field containing the length of the keyword entry. The keyword entry is made up of positional parameter entries describing the processing to be done on the positional parameters associated with the keyword.

Each parameter entry contains two kinds of information. Length and error checking information is followed by control information, which describes the functions to be performed on the parameter, and the location in which the result is to be stored.

The first byte in each parameter entry (the parameter PDT length field) contains the length of the entry; the first half of the second byte (the control field length field) contains the length of the control information. The format of the remainder of the entry depends on the type of parameter and on the functions to be performed.

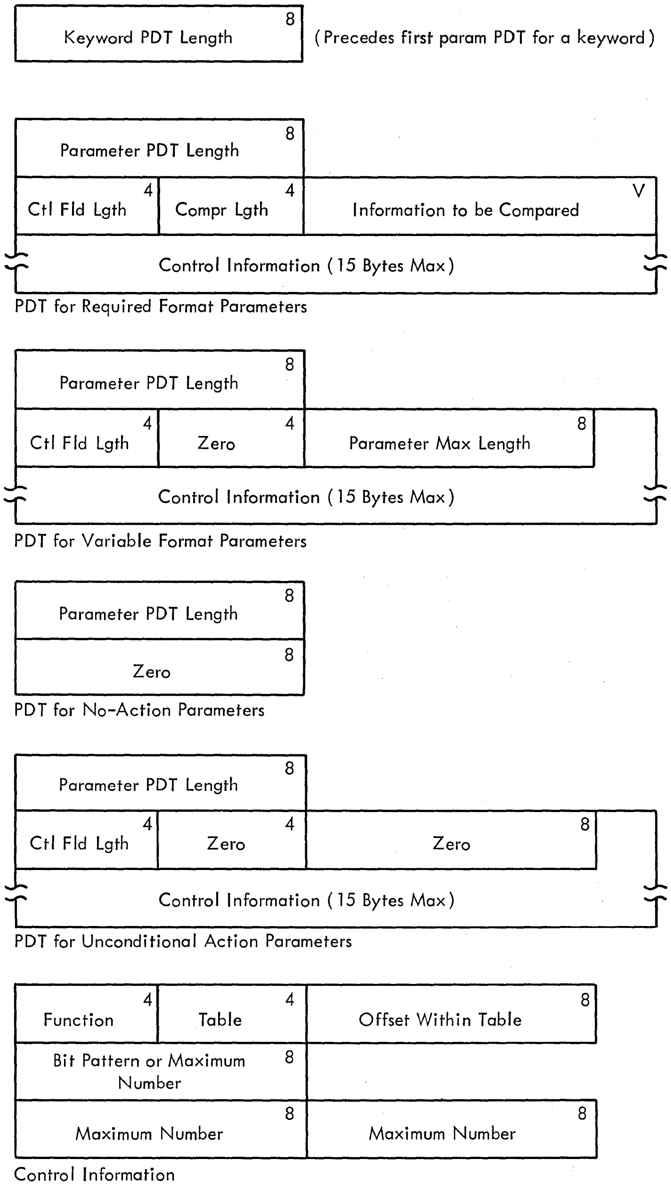

Keyword PDT Length 8
(Precedes first param PDT for a keyword)
Parameter PDT Length 8
Ctl Fld Lgth 4
Compr Lgth 4
Information to be Compared V
Control Information (15 Bytes Max)
PDT for Required Format Parameters
Parameter PDT Length 8
Ctl Fld Lgth 4
Zero 4
Parameter Max Length 8
Control Information (15 Bytes Max)
PDT for Variable Format Parameters
Parameter PDT Length 8
Zero 8
PDT for No-Action Parameters
Parameter PDT Length 8
Ctl Fld Lgth 4
Zero 4
Zero 8
Control Information (15 Bytes Max)
PDT for Unconditional Action Parameters
Function 4
Table 4
Offset Within Table 8
Bit Pattern or Maximum Number 8
Maximum Number 8
Maximum Number 8
Control Information

Figure 74. Parameter Descriptor Table (PDT)

There are four types of parameters:

- A required-format parameter is a known string of characters. The first positional parameter following the DISP= keyword, for example, must be either "OLD", "NEW", "MOD", or "SHR". In this case, since there are four possibilities, there are four parts to the entry; the Test and Store routine compares the parameter to the constant in each of the four parts, and performs the function specified in the control information field of the part in which it obtained an equal compare.

- A variable-format parameter may be any string of characters up to a known maximum length. The classname parameter of the SYSOUT keyword is an example; since there are 36 system output class names permitted in the system, a series of comparisons would be unwieldy. The compare length byte in such an entry is zero; the third byte in the parameter entry specifies the maximum number of digits allowed.

- A no-action parameter specifies a default option. The MSGLEVEL keyword, for example, may be omitted by the programmer, in which case MSGLEVEL=2 is assumed. If the MSGLEVEL keyword is present, however, the positional parameter must be either 0, 1, or 2. If MSGLEVEL=0 or 1 is specified, the bit must be turned on, but if MSGLEVEL=2 is specified, no action need be taken.

- An unconditional-action parameter indicates that the presence of the parameter requires that the same functions be performed regardless of the form or content of the parameter. When the SUBALLOC keyword is encountered, for example, certain switches must be set, regardless of how much or what kind of space has been requested.

The control information portion of a parameter PDT entry defines the operations to be performed when the parameter is processed, specifies the location in which theresults are to be stored, and may contain data to be used in the operation. The control information portion may be up to 15 bytes in length; it consists of the following fields:

- Function: The first four bits of a control information field contain a number from 0 to 7, which specifies one of the following operations:

  - OR (Code 0): A logical or operation is performed, using the bit pattern field in the control information portion of the entry, against the bit pattern at the location specified by the table and offset fields.

  - CVB1 (Code 1): A convert to binary operation is performed and a maximum value check is made. The converted information is stored (right justified) in the one-byte field specified by the table and offset fields, and compared against the maximum value, which is right-justified in the third byte of the control information part of this entry.

  - CVB2 (Code 2): This operation is similar to CVB1, except that the result is right-justified in a two-byte field, and the maximum value is found right-justified in the third and fourth byte of the control information portion of the entry.

  - CVB3 (Code 3): This operation is similar to the CVB1 and CVB2 operations, except that the result is right-justified in a three-byte field, and the maximum value is found in the third, fourth, and fifth byte of the control information portion of the entry.

  - AND (Code 4): A logical and operation is performed, using the bit pattern field in the control information portion of the entry against the bit pattern at the location specified by the table and offset fields.

  - MVC (Code 5): A move characters operation is performed, using the parameter length specification in the internal text buffer. The parameter is moved to the location specified in the table and offset fields in the entry.

  - First Character Alpha Check and MVC (Code 6): This function is similar to the MVC function, except that before the move is performed the first character of the parameter is inspected to insure that it is alphabetic.

  - Alpha/Numeric Check and MVC (Code 7): This function is similar to the MVC function, except that before the move is performed a character (a one character parameter) in the text buffer is inspected to determine whether it is alphabetic or numeric.

- Table: The second four bits of the control information portion of a parameter PDT entry contains a number between 0 and 15 that specifies the table in which the result of the operation is to be stored. Figure 75 lists these tables and the corresponding numbers.

- Offset: The second byte of the control information of an entry contains the offset, from the beginning of the table, of the field in which the results of the operation are to be stored.
- Bit Pattern/Maximum Number: The third through fifth bytes of the control information portion of the entry are used for those operations that require data for logical or comparison functions. If the operation is AND or OR, the third byte contains the bit pattern. If the operation is a CVB operation, the third, fourth and fifth bytes contain the binary representation of the maximum value allowed for that parameter.

Code Number	Table
0	Local Work Area (LWA)
1	Job Control Table (JCT)
2	Step Control Table (SCT)
3	Job Account Control Table (ACT)
4	Step Input/Output Table (SIOT)
5	Job File Control Block (JFCB)
6	JFCB Extension (JFCBX)
7	Volume Table (VOLT)
8	Data Set Name Table (DSNT)
9	Refer-back Dictionary 1
10	Refer-back Dictionary 2
11	Procedure Override Table
12	Step Account Control Table (ACT)
13	Reserved
14	Reserved
15	Interpreter Work Area (IWA)

Figure 75. Table Codes Used in the Test and Store Subroutine

The Dictionary Entry Routine

The Dictionary Entry routine (module IEFVGI) is used by the EXEC statement processor routine and the DD statement processor routines to place an entry in the refer-back dictionary (Figure 80).

The dictionary is maintained in a 176-byte area in the IWA; if the dictionary overflows the area, it is written out to the job's input queue entry, a new dictionary is initialized in the IWA, and the new dictionary is chained to the previous copy in the queue entry.

The Dictionary Search Routine

The Dictionary Search routine (module IEFVGS) is used by the EXEC and DD Statement Processor routines to search the refer-back dictionary for the address of a previously defined SCT, SIOT or JFCB. It returns control to the calling routine with a pointer to the dictionary entry containing the address (in the queue data set) of the required table.

The Interpreter Message Routine

The Interpreter Message routine (module IEFVGM) is used by the Interpreter Control routine and JCL Statement Processor routines when a JCL statement or diagnostic message must be placed in an SMB, and to enqueue SMB's and an SCD for each job. It also reserves space in the first SMB, which the Job Termination Exit routine (module IEFVHN) may fill with a message indicating the SYSOUT classes that have data in them.

The Queue Manager Interface Routine

The Queue Manager Interface routine (module IEFVHQ) is used by those Interpreter routines that need to assign space, and to read and write records in the queue. It provides a queue manager parameter area, and passes control to the queue manager to perform the function specified by the calling routine. On the return from the queue manager, it resets the parameter area so that it specifies an assign and write one record operation, and returns control to the caller.

The Record Compression Routine

The Record Compression routine (module IEZNCODE) is used by the In-stream Procedure routines to compress JCL statements so that they will occupy less queue space.

The routine compresses statements by replacing each series of occurences of a character specified by the caller with a count field. The count field contains a one- or two-byte hexadecimal number, the left half of which is the number of occurrences of the specified character in the uncompressed series and the right half of which is the number of other characters preceding the series. The high order bit of the count field indicates whether the count field is one or two bytes long: if the bit is on, the count field is one byte long; if it is off, the count field is two bytes long.

The routine also automatically blocks the compressed records in blocks whose size is specified by the caller. To mark the end of a block, the routine stores an end-of-block indicator X'0000' following the last field of the last compressed record in the block.

The entire compression technique is illustrated by Figure 76. The uncompressed record

//A JOB

in Figure 76 is compressed by the routine as follows:

- The first field of the compressed record contains the length of the compressed record.

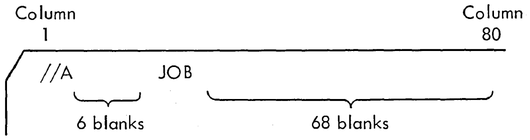
- The next field is a count field replacing the first series of blanks in the record. The "E" in X'E3' is the number of blanks, plus eight because the high order bit is on to indicate a one-byte count field. The "3" is the number of nonblank characters ("//A") preceding the series of blanks.

- The next field contains the nonblank characters "//A".

- The next field is a second count field, which replaces the second series of blanks in the record. The "44" in X'4403' is the number of blanks in the last series in the record. The "03" indicates that there are three nonblank characters ("JOB") preceding the series of 68 blanks.

- The next field contains the three nonblank characters "JOB".

- The last field contains the end-of-block indicator X'0000' that follows the last record in a block.



X '000B'	X 'E3'	//A	X '4403'	JOB	X '0000'

Figure 76. Record Compression Technique

This compression technique is performed whenever the Record Compression routine receives control. When the routine is entered, it is passed the address of a compression parameter area (see Figure 77) containing the address of the record to be compressed and the address where the compressed record is to be stored. The parameter area also contains the length of the uncompressed record, the maximum length of the compressed record, and the character to be replaced. Also included in the parameter area is a flag field which the caller may use to identify a special record. The high order bit of the field is the flag and the other bits must be zero. To flag a record, the caller sets the bit in the parameter area. The routine will set the high order bit in the length field of the correspoonding compressed record.

The routine determines from the parameter area the address of the record to be compressed, scans and compresses the record, and stores the compressed version, if it is not too long, in the location specified in the parameter area. If the compression process completes normally, the routine updates the compressed record address to the byte following the current compressed record and stores the end-of-block indicator at the address. If the record is not the last one in the block, the end-of-block indicator will be overlayed by the next record. Then the routine returns control to the caller with a return code of zero. If the compressed record would have exceeded the specified maximum length, the routine returns control with a return code of four.

The Record Decompression Routine

The Record Decompression routine (module IEZDCODE) is used by the In-stream Procedure routines to decompress records that were previously compressed by the Record Compression routine. To decompress the compressed records, the routine merely reverses the compression process.

When it is invoked to decompress a record, the Record Decompression routine is passed the address of the parameter area shown in Figure 77. The parameter area contains the address of the location where the decompressed record is to be stored, the address of the compressed record, the length of the decompressed record and its maximum length, and the character to be inserted. It also contains the flag field that may be used by the caller to identify a special record. If the high order bit of the length field in the compressed record is on, the routine sets the high order bit of the flag field and ensures that the other bits of the field are zero.

<table>
<tr><td colspan="3">Address of Uncompressed or Decompressed Record 4</td></tr>
<tr><td colspan="3">Address of Compressed Record 4</td></tr>
<tr><td colspan="2">Length of Uncompressed or Decompressed Record 2</td><td>Maximum Length of Output Record 2</td></tr>
<tr><td>Omitted Character 1</td><td>Special Record 1</td><td>Reserved 2</td></tr>
</table>

Figure 77. Compression/Decompression Parameter Area

The routine finds the address of the compressed record from the parameter area, decompresses the record, and stores it in the specified location, if it is not too long. If the decompression process completes normally, the routine places the address of the next compressed record in the decompression parameter area and performs deblocking if necessary. Then it returns control to the caller with a return code of zero. If the decompressed record would have been longer than the specified maximum length, the routine returns with a return code of eight. If the routine scans an end-of-block mark instead of a compressed record, it returns with a return code of four.

INTERPRETER EXITS

There are five exits from the interpreter: an Accounting routine exit, an input access method exit, two SMF exits, and a Return routine exit. The use of these exits is controlled by the use of the exit list (see Figure 64); the linkage is shown in Figure 78.

The Accounting routine exit may be used to pass control from the Job Statement Processor, DD Statement Processor, and Scan routines whenever a complete statement has been accumulated. When the Exit routine is given control, register 1 contains a pointer to a four-word parameter list. The parameter list contains pointers to the internal text buffer, the QMPA for the input queue entry, the Queue Management routines' entry point, and the pointer in the third word of the NEL.

Generally, the third word of the NEL contains a pointer to the JCL. However, when the System Task Control routine invokes the interpreter, it places in the third word of the NEL a pointer to a two-word parameter list. The first word of this parameter list points to the JCL used by the Internal JCL Reader routine (module IEEVRJCL); the second word of the parameter list points to a work area used for communication between the System Task Control Post Scan Exit routine (module IEEPSN) and the Interpreter Control routine (module IEEVRCTL).

The Input Access Method routine exit may be used to specify a special Access routine for reading the input stream. If no such specification is made, or if the default option is taken, the interpreter open module issues the OPEN macro instruction to open the input stream DCB. If a specification is made, the routine stores the address of the specified routine in the DCB, so that subsequent GET macro instructions will cause the special routine

<table>
<tr><th colspan="3">Accounting Routine Exit (module IEFVFA)</th></tr>
<tr><th>Definition</th><th>Interpreter Action</th><th>User Return</th></tr>
<tr><td>Default</td><td>BALR to V-type address constant IEFACT</td><td>RETURN</td></tr>
<tr><th colspan="3">Input Access Method Routine Exit (module IEFVHA)</th></tr>
<tr><th>Definition</th><th>Interpreter Action</th><th>User Return</th></tr>
<tr><td>Default</td><td>OPEN IEFRDER DCB</td><td>------</td></tr>
<tr><td>Address</td><td>Store specified address in IEFRDER DCB</td><td>RETURN</td></tr>
<tr><td>Name</td><td>Store corresponding address in IEFRDER DCB</td><td>RETURN</td></tr>
<tr><td>V-type address constant</td><td>Invalid</td><td>------</td></tr>
<tr><th colspan="3">Return Routine (module IEFVHN)</th></tr>
<tr><th>Definition</th><th colspan="2">Interpreter Action</th></tr>
<tr><td>Default</td><td colspan="2">Branch on register 14</td></tr>
<tr><td>Address</td><td colspan="2">Branch to specified address</td></tr>
<tr><td>Name</td><td colspan="2">XCTL to specified name</td></tr>
<tr><td>V-type address constant</td><td colspan="2">Branch to V-type address constant IEFK3</td></tr>
</table>

Figure 78. Interpreter Exit Linkage

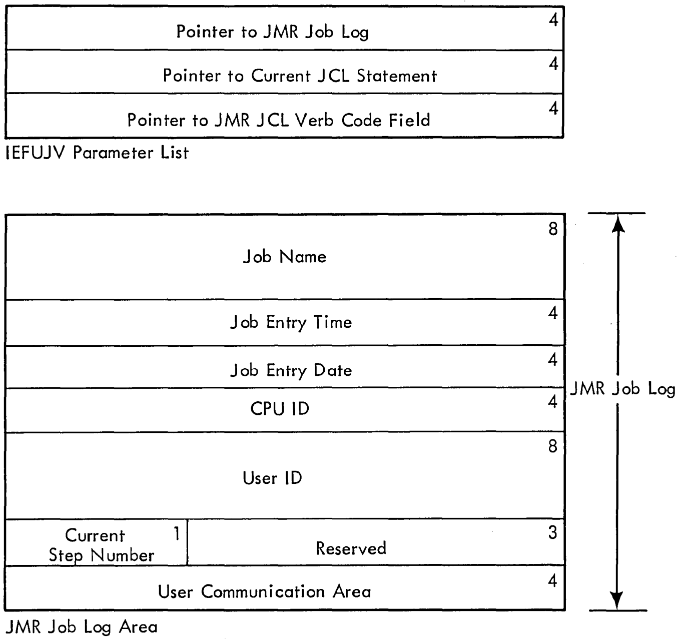
to be executed. Address information for the special access method may be passed to the interpreter in the JCL pointer field of the interpreter entrance list.

There are two SMF exits in the Interpreter. The JCL Validation exit is taken each time a JCL statement has been read (but before it is scanned), and the JCL Enqueue exit is taken immediately before a job is enqueued. In both cases, control is passed to the user's routine with register 1 pointing to the parameter list shown in Figure 79. The user's routine, which must be named IEFUJV, returns control to its caller with a return code of 0 (continue processing) or 4 (cancel the job) in register 15.

The interpreter returns control to its caller when an end-of-data condition is encountered in the input stream, or when the stop ECB has been posted. If no Return routine specification has been made, or if the default option has been taken, the interpreter returns to its caller by branching on general register 14. Otherwise, the interpreter will branch as directed by the exit list specification.

INTERPRETER TERMINATION

At end-of-data in the input stream, or when the interpreter determines that a STOP command has been issued, control is passed to the Interpreter Termination routine (module IEFVHN). This routine closes the procedure library PDS, and unless a special access method was specified, it closes the input stream data set. The routine updates the interpreter entrance list with JCL, JCT, and QMPA pointers as required, then determines whether a JCL Validation routine was loaded. If so, it issues the DELETE macro instruction; in any case, it releases the main storage obtained for the DSENQ table, the two DCB's, the IWA, the LWA and the JMR. When processing is complete it issues a closing message to the requesting operator, restores general registers to their value upon entry to the interpreter (at module IEFVH1), loads register 1 with a pointer to the entrance list, and returns control to the caller of the interpreter.


Pointer to JMR Job Log
4
Pointer to Current JCL Statement
4
Pointer to JMR JCL Verb Code Field
4
IEFUJV Parameter List

JMR Job Log Area

Figure 79. IEFUJV Parameter List and JMR Job Log

Fixed Portion:

3	1	3
Dictionary Queue Address	Table ID	Chain Pointer or Zeros

Type 1 Entry: EXEC PROC

1	1	1-8
Length	Indicator X'80'	Name

Type 2 Entry: EXEC PGM

1	1	3	1-8
Length	Indicator X'40'	Queue Address of SCT	Name

Type 3 Entry: EXEC DD

1	1	3	3
Length	Indicator X'10'	Queue Address of SIOT	Queue Address of JFCB
			1-8
Name			

Type 4 Entry: DD DCB

1	1	3	3
Length	Indicator X'08'	Queue Address of SIOT	Queue Address of JFCB
			1-8
Name			

Type 5 Entry: EXEC PGM (within a procedure)

1	1	3	1-8
Length	Indicator X'20'	Queue Address of SIOT	Name

Figure 80. Refer-Back Dictionary

I/O Device Allocation

The I/O requirements of job steps (and of system tasks such as the reading, writing, and MOUNT command tasks) are specified in DD statements, which are included in input streams. Each DD statement is an I/O request; it specifies the attributes, and the device and volume requirements, of a data set. The interpreter reads these requests, translates them into tabular format, and places the tables in the input queue. The tables are the input data for the I/O Device Allocation routine (Figure 81), which is used as a subroutine by the initiator to perform the following functions:

- Allocation housekeeping is performed in preparation for the allocation process. Since, in a multitasking system, the unit control blocks (UCBs) are vulnerable to multiple references, the Allocation Entry routine uses the ENQ macro instruction to prevent access by other tasks to the UCBs. This ensures that the termination or allocation of another task cannot take place involving units and volumes required by allocation for this task. The Allocation Entry routine also obtains a system message block (SMB) for allocation messages. On the first step of the job (if a DISPLAY JOBNAMES command has been issued) it displays the job name. Condition codes are processed, and if they indicate that the step is not to be run, subsequent processing is performed in the flush mode; a dummy TIOT is constructed, an error code is returned to the caller, and the step is not executed.

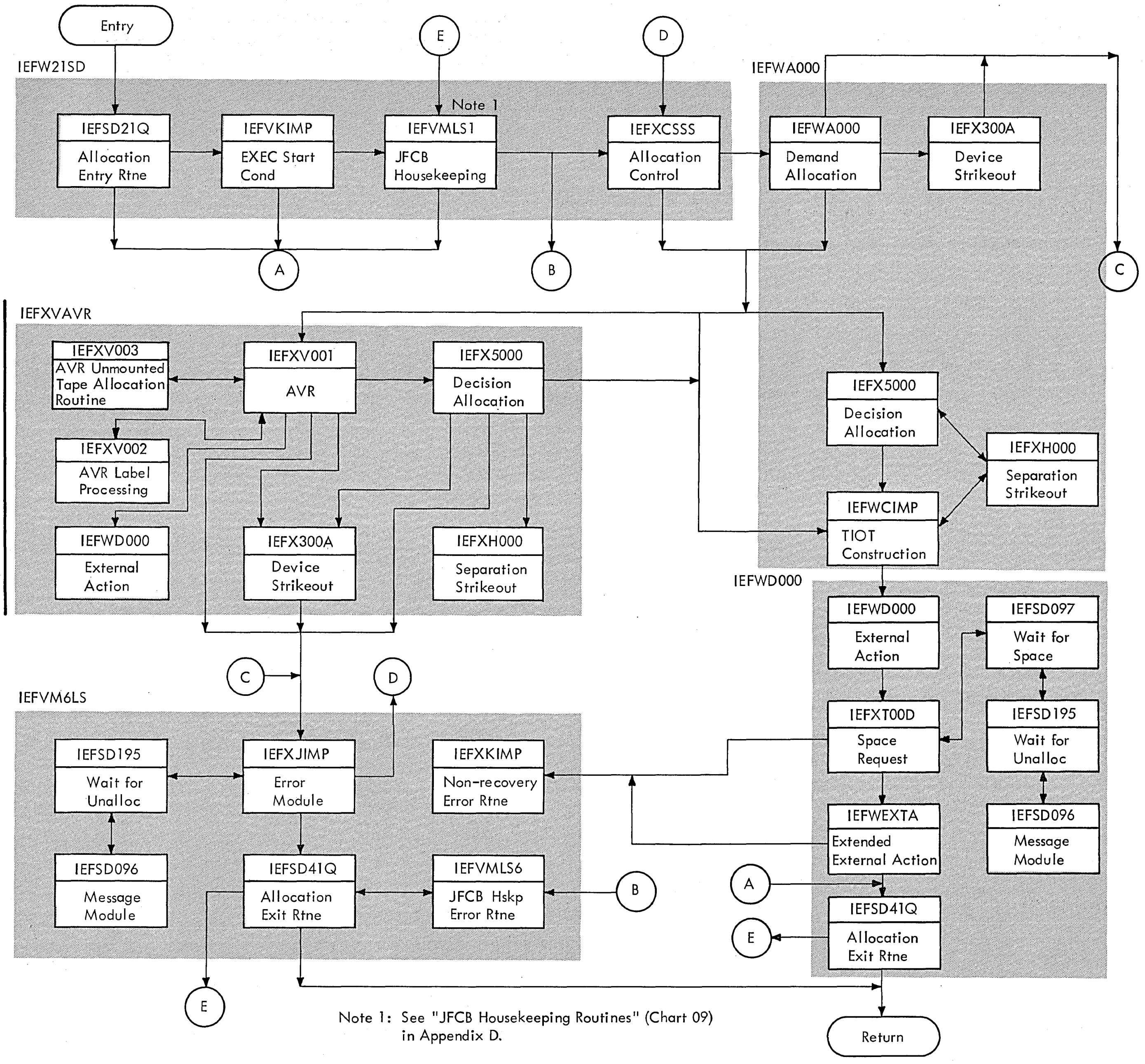
- Gathering information on which to base unit and volume assignments is performed by the JFCB Housekeeping routine, the Allocation Control routine, and the Demand Allocation routine. When an allocation request does not explicitly specify volume or unit information, the information is obtained and added to existing tables. The main storage requirements for other allocation tables are calculated, the storage is obtained, and the tables are constructed. As table construction is taking place, volume affinity requests are resolved, and the device requirements of each data set are calculated.

- Unit assignment is essentially a process of elimination. The Demand Allocation routine first assigns units to those data sets requiring resident and reserved volumes, then, for each unallocated data set, it eliminates the obviously ineligible units. If this process has reduced the number of units eligible for allocation to a data set to the number required, the units are allocated, as are units that have been specifically requested.

  The Decision Allocation routine then processes separation requests, eliminating the units that violate either unit separation or channel separation requests. Next, specific volume requests are processed: requests for volumes mounted on eligible devices are satisfied, then the data set having the most restrictive requirements is selected, and a unit is allocated to it. This process is repeated until all requests (except requests for space on public volumes) have been honored.

- TIOT construction involves calculating the size of the task input/output table (TIOT), constructing entries for previously allocated data sets, and for data sets that require public volume space. Since the public volume requests have not had units assigned to them, the TIOT Construction routine begins the process of allocation by eliminating ineligible units, and placing, in each entry, a list of units that are eligible for allocation to that data set. The eligible units are listed in order of suitability; the most suitable is assumed to be allocated, but the allocation is not actually completed until direct access space has been obtained.

- Space assignment is performed in the External Action, Space Request, DADSM Error Recovery, TIOT Compression, and Extended External Action routines. The External Action routine issues mounting instructions to the operator for all volumes needed by the task. When the operator mounts the volumes that contain new data sets, the routine verifies that the correct direct access volumes have been mounted. Verification of volumes that contain old data sets is delayed until the Extended External Action routine receives control at the end of allocation. (All requested data cell volumes are verified by the External Action routine.) The Space Request routine then uses the Direct Access Device Storage Management (DADSM) routines to obtain space for new data sets on both private and public direct access volumes. If the space request is not honored, the DADSM Error Recovery routine determines what action is to be taken. The allocation to data sets that require public volume space



Note 1: See "JFCB Housekeeping Routines" (Chart 09) in Appendix D.

Figure 81. I/O Device Allocation Routine

is completed, and the unused units are removed from the lists in the TIOT entries. The TIOT Compression routine compresses the TIOT to its final size, places allocation messages in SMBs, and passes control to the Extended External Action routine. The Extended External Action routine verifies that the correct direct access volumes have been mounted in response to requests for volumes that contain old data sets. Then the routine passes control to the Allocation Exit routine.

- Allocation exit processing writes out the SMB containing allocation messages, and uses the DEQ macro instruction to release the interlock on access to the UCBs before returning control to the caller. If an unrecoverable error was encountered during allocation, control is passed to the JFCB Housekeeping routine (see Gathering Information) and flush mode processing is initiated.

The functions described above are performed by a set of modules called the I/O Device Allocation routine. Since similar processing is required at several points in the process of allocation, the modules use several routines in common. The common routines (which are discussed in detail at the end of this section) are:

- The Allocation Recovery routine, which attempts to make additional devices available when a lack of available units or direct access space has made allocation impossible. If SMF is included, the addition of available devices must result in the writing of a Type 10 allocation recovery record to the SMF data set.
- The Device Strikeout routine, which completes the allocation of a selected unit in the allocation work tables, and, where appropriate, makes it unavailable to other data sets.
- The Separation Strikeout routine, which removes channel and unit separation request violators from consideration as eligible units.
- The Unsolicited Device End Posting routine, which posts the ECB when the appropriate device becomes ready.

ALLOCATION HOUSEKEEPING

Before I/O device allocation can begin, certain housekeeping functions must be performed. These functions are performed by the Allocation Entry routine (module IEFSD21Q) and the EXEC statement Condition Code Processor routine (module IEFVKIMP).

If the task uses the SMF option, module IEFSD21Q issues a TIME macro instruction to obtain a time stamp that indicates when device allocation started. The module then stores the time stamp in the TCT.

Protecting UCB Information

The I/O Device Allocation routine and the Termination routine, both of which modify UCBs, may operate concurrently while performing separate system tasks. Therefore, multiple references to UCBs are possible, and must be prevented.

The Allocation Entry routine (module IEFSD21Q) first determines if Dynamic Device Reconfiguration (DDR) is in the system.

Then, to protect UCB information, the routine issues an ENQ macro instruction (using the exclusive option) that refers to a name that represents access to the UCBs by the I/O Device Allocation routine. Because the ENQ macro instruction is located at the beginning of the I/O Device Allocation routine, the effect is to prevent allocation from being performed for more than one job step at a time (by routines performing other tasks).

At this point, if DDR was found in the system, the Allocation Entry routine stores its TCB; if TSO was found in the system, the Allocation Entry routine also stores the TJID for that TCB into IORMSCOM. The routine then issues a WAIT macro instruction for the ECB that is posted except during DDR processing. When DDR has completed processing the ECB is posted again. This prevents the concurrent execution of Allocation and DDR. (See the Input/Output Supervisor PLM for a discussion of DDR processing.)

When the JFCB Housekeeping routines receive control, module IEFVMLS1 issues an ENQ macro instruction that refers to a name representing access to the UCBs by the Termination routine and the SYSOUT writer. This prevents termination of any job step while JFCB housekeeping is taking place. The routine also issues an ENQ macro instruction to prevent Allocation routines for another task from using the UCBs during this allocation processing. Similarly, when the Demand Allocation routine (module IEFWA000) receives control, it also issues an ENQ macro instruction to prevent access of the UCBs by the Termination routine and the SYSOUT writer during its processing.

When the JFCB Housekeeping routines pass control to the Allocation Control routine (module IEFXCSSS), it issues a DEQ macro instruction to allow access of the UCBs by the Termination routine and the SYSOUT writer while the Allocation Control routine is building tables.

If allocation for this task is impossible because of a lack of devices, volumes, or space, the Wait for Unallocation routine issues two DEQ macro instructions to allow the Termination routines, the SYSOUT writer, and the Allocation routines for another task to access the UCBs. When a task terminates and its I/O devices are released, control returns to the Wait for Unallocation routine, which issues two ENQs for the UCBs, and the I/O Device Allocation routine attempts to allocate with the additional devices.

When allocation is complete, the Allocation Exit routine issues three DEQ macro instructions, allowing other allocation and Termination routines to refer to the UCBs.

Obtaining an SMB

The Allocation Entry routine obtains main storage for a system message block (SMB), and uses the Queue Management Read routine to bring the last SMB created by the interpreter (for the current step) into main storage. During the process of allocation, allocation messages for the programmer are placed in this SMB. After allocation is complete, the Allocation Exit routine places the SMB in the appropriate output queue.

Displaying the Job Name

If the current step is the first step of the job, the Allocation entry routine tests the job and time notification bits in the master scheduler resident data area. These bits are set on as a result of either the MONITOR JOBNAMES or MONITOR SESS (for foreground sessions only) command, and set off as a result of the STOP JOBNAMES command[1]; if the job notification bit is on, and the job termination status bit in the LCT is off, the routine uses the WTO or TPUT macro instruction to issue a job started message to the requesting operator or TSO user. If the time notification bit is also on, the routine includes the time in the message.

The job and time notification bits are also tested in the Job Termination Exit routine of the Termination routine, which issues a message to notify the requesting operator that the job has ended.

Job and Step Flush

If a job is canceled because of JOB statement condition codes, operator command, or unrecoverable errors encountered by the interpreter or initiator, the remaining steps in the job are not executed.

The Allocation Entry routine tests the job-failed and job-flush bits to determine whether the job has been canceled. If so, EXEC statement condition code processing is unnecessary (it is already known that the step will not be run) and control is passed to the JFCB Housekeeping Control routine. The job-failed bit is set off; the job-flush bit is set on to specify flush mode processing.

If the job is not canceled, control is passed to the EXEC statement condition code processor.

When a job, canceled as previously indicated, is in the flush/fail mode, the extent to which the system performs allocation processing for subsequent job steps depends on both the occurrence of and the condition of SYSIN data sets, in the following manner:

- If there are no existing SYSIN data sets for the subsequent job steps, the system terminates the job immediately.

- If a mounted SYSIN data set exists for one or more job steps, the system performs allocation processing as necessary to delete the SYSIN data sets.

- If there is one or more unmounted SYSIN data sets for the job (as a result of unsatisfied mount requests), the system performs allocation processing to permit a user one more chance to satisfy the mount request(s). If the mount requests still are not satisfied, the system terminates the job without deleting the unmounted SYSIN data sets.

EXEC Statement Conditional Execution Processing

If the programmer uses the COND keyword in an EXEC statement, and if COND=ONLY or if the step is not the first step of the job, the EXEC Statement Conditional Execution routine (module IEFVKIMP) determines whether the step should be bypassed or run. Otherwise, the routine passes control to the JFCB Housekeeping routine.

The EXEC Statement Conditional Execution routine tests the ABEND indicator in the JCT to determine whether any steps have been abnormally terminated, and it tests the high-order bit of the eighth step condition code field in the SCT to determine whether COND=EVEN or COND=ONLY is specified. The step will immediately be bypassed if:

- One or more previous steps have been abnormally terminated, and the COND field specifies neither EVEN nor ONLY.
- No steps have been abnormally terminated, and the COND field specifies ONLY.

If neither of these conditions exists, but the COND field specifies condition codes and operators, the routine uses the Queue Management Read/Write routine to bring the SCT of the step specified in the first COND field into main storage. The return code issued by that step (in field SCTEXEC) is then compared to a condition code specified for the current step, and the result is matched to the specified condition code operator.

----

[1] These bits are also set off if the execution of a VARY command places all consoles receiving the display in an inactive or offline status.

The process is repeated for each COND parameter (if no step name was specified, the process is repeated for each preceding step) until either the result of the comparison matches the condition code operator, or until all codes and their operators have been tested. If the result of the comparison matches the condition code operator, the condition has been met, and the step is bypassed.

When a step is bypassed, the EXEC Statement Conditional Execution routine loads an error code and passes control to the Allocation Exit routine. The Allocation Exit routine sets the step-flush bit on, passes control to the JFCB Housekeeping routine, and the step is processed in the flush mode.

If the step is to be executed, the EXEC Statement Conditional Execution routine passes control to the JFCB Housekeeping routine.

GATHERING INFORMATION

When the housekeeping functions have been performed, the information necessary to the allocation process is gathered and stored in tabular format for easy reference by the allocation routines. Volume information is added to existing tables by the JFCB Housekeeping routines. The Allocation Control routine then calculates the sizes of the new tables that are required, and obtains storage for them. The tables are constructed in the first part of the Demand Allocation routine, while volume affinity requests are being resolved, and device requirements calculated.

Completing Tables

The JFCB Housekeeping routines complete volume information within the following tables, in preparation for their use in allocating I/O devices:

- Job file control block (JFCB)
- Step input/output table (SIOT)
- Step control table (SCT)
- Volume table (VOLT)

In order to enter information in these tables, reference may be made to the system catalog, via the LOCATE and OBTAIN macro instructions. Passed data sets are also processed; a passed data set queue (PDQ) is constructed, and entries are made for the first reference to each data set. Subsequent references cause the existing entries to be updated.

Table completion and construction is performed in the following (JFCB Housekeeping) routines:

- JFCB Housekeeping Control routine (module IEFVMLS1)
- Dedication Determination routine (module IEFSD180)
- Allocate Processing routine (module IEFVMLS1)
- Direct System Output Determination routine (module IEFDSOAL)
- Fetch DCB Processing routine (module IEFVM2LS)
- GDG Single Processing routine (module IEFVM3LS)
- GDG All Processing routine (module IEFVM4LS)
- Patterning DSCB Processing routine (module IEFVM5LS)
- Error Message Processing routine (module IEFVM6LS)

JFCB Housekeeping Control Routine

Module IEFVMLS1 is the entry and exit module for the JFCB Housekeeping routines. Upon entry, module IEFVMLS1 issues an ENQ macro instruction to prevent access to the UCBs by the Termination routine. Module IEFVMLS1 also issues an ENQ macro instruction to prevent Allocation routines from using the UCBs to allocate another task. Then the Control routine (which is part of IEFVMLS1) uses the Queue Management Read routine to bring each SIOT into main storage, examines the SIOT, and depending on the types of processing required, it passes control, in turn, to the appropriate Processing routines. When an SIOT has been processed, and written back into the input queue, the next SIOT is read; when the last SIOT for the step has been processed, control is passed to the Allocation Control routine.

Flush Mode Processing: If the job-flush bit or step-flush bit is on, the step is to be processed in flush mode. Since the step is not to be run, all data sets except system input data sets and OLD system output data sets are processed as dummies, no volume information will be needed, and the only processing required for system input data sets is to set the dummy bit on in all other SIOTs. For an OLD system output data set, during system restart processing, the volume table will be updated with volume information from the

JFCB. The SCT will be updated to reflect the increase in size of the volume table. When the last SIOT has been written back into the input queue, control is passed to the Allocation Control routine as in normal processing.

Allocate Processing Routine

The JFCB Housekeeping Control routine branches to the Allocate Processing routine (which is also part of module IEFVMLS1) under the following circumstances:

- The DD statement refers, by data set name, to a passed data set or a cataloged data set.
- The DD statement uses ddname or stepname.ddname to refer to a data set described in a previously processed DD statement.

In addition, when a unit name is specified in a DD statement, the Allocate Processing routine converts it to unit type through use of the device name table (DNT). When a unit address is specified in a DD statement, the routine searches the UCBs for the one that corresponds to the EBCDIC unit address in the SIOT of the requested unit. When the correct UCB is found, the device type and the UCB address are entered into the unit type field of the SIOT.

When the data set reference is a data set name, the PDQ is examined. If it contains an entry for the data set referred to, the SIOT and JFCB are read into main storage; they furnish the volume and device information for the subject data set.

If, however, there is no PDQ entry for the data set referred to, a LOCATE macro instruction is used to find the data set in the catalog. The volume control block or data set entry is then used to complete the volume and device information.

If the LOCATE macro instruction return code indicates that the required data set is cataloged on an unmounted control volume, IEFMCVOL is entered to cause allocation of a device for the required volume and request that the operator mount it. If the control volume can be mounted, which depends upon device and volume availability, control is returned to the JFCB Housekeeping routine. (If the control volume cannot be mounted, an abnormal system termination occurs.) The LOCATE macro instruction is issued again, the control volume is unallocated, and further requests for the step are allocated.

When a data set resides on more than five volumes, a JFCB extension block is created; the additional volume and device information is stored in it, and it is added to the input queue entry.

When the reference is to a previously processed DD statement (by ddname or stepname.ddname), a check is made to determine whether the DD statement appeared in the step being processed. If so, the SIOT and JFCB associated with the data set referred to are read into main storage and the volume and device information extracted. If the DD statement appeared in a previous step of the job being processed, the SIOT and JFCB constructed for that step are read in, and the volume and device information extracted.

Dedication Determination Routine

The JFCB Housekeeping Control routine branches to the Dedication Determination routine if the DD statement being processed requests a dedicated data set. A request is made if the SCTANAME field of the SIOT is not all blank, therefore indicating DSNAME=&NAME.

The Dedication Determination routine checks the following parameters of the DD statement to ensure that they pertain to a dedicated data set:

- The DSORG Parameter must indicate other than an ISAM data set.
- The DSNAME parameter must match a DDname in the Initiator's TIOT.
- The "units" subparameter of the SPACE parameter must specify average block length. The "quantity" and "increment" subparameters of the SPACE parameter must reflect a total size not larger than the data set whose DDname was matched in the Iniatator's TIOT. The "directory" subparameter of the SPACE parameter must not be larger than that of the data set whose DDname was matched in the Iniator's TIOT.

If any one of these conditions is not satisfied, control is returned to the JFCB Housekeeping Control routine.

If all the conditions are satisfied, the Dedication Determination routine does the following:

- Ensures that there will be no checkpoints and no restarts by forcing NC and NR in the SCT.
- Ensures that the dedicated data set will not be deleted by forcing DISP=OLD, KEEP in the SIOT.

- Copies volume and device information and the DSNAME from the SIOT and JFCB of the dedicated data set into the SIOT and JFCB created from the DD statement requesting the dedicated data set.

- Updates the volume table and the PDQ. Control is then returned to the JFCB Housekeeping Control routine.

Direct System Output Determination Routine

The JFCB Housekeeping Control routine branches to the Direct System Output (DSO) Determination routine (module IEFDSOAL) when it encounters an SIOT that describes a system output data set.

When it is entered, the DSO Determination routine issues an ENQ macro instruction (in the shared mode) specifying the chain of direct system output control blocks (DSOCBs). The routine scans the chain for DSOCBs assigned to the job; if there is no DSOCB for the output class specified in the SIOT, the routine issues a DEQ macro instruction and returns control to the JFCB Housekeeping control routine.

If there is a DSOCB for the class specified in the SIOT, the DSO Determination routine sets all bits off in the SIOT data set characteristics field except the SYSOUT and NEW bits, which it sets on. It stores the three-character unit address of the DSO unit in the SIOT unit address field, and uses queue management to read the JFCBs created for the data set and for the DSOCB into main storage.

When the JFCBs are in main storage, the DSO Determination routine updates the JFCB for the data set with information from the JFCB created for the DSOCB. It uses queue management to write the data set JFCB back into the queue entry.

The DSO Determination routine then sets the active bit on in the DSOCB and inspects the job separator and message class bits in the DSOCB. If the job separator bit is off, or if the message class bit is on, the routine sets the DSO work bit on in the SCT to indicate that job separator or message processing is required.

Finally, the DSO Determination routine issues a DEQ macro instruction specifying the DSOCB chain and returns control to the JFCB Housekeeping Control routine.

Fetch DCB Processing Routine

Module IEFVM2LS completes volume and device information when the data set referred to contains a program that was created in a previous step and is to be executed as the current step.

GDG Single Processing Routine

Module IEFVM3LS obtains the data set name of a generation data group (GDG) member and completes volume and device information entries for the member. If this is the first reference (in the current job) to the GDG, the routine constructs a GDG bias count table; if previous references have been made, the table is updated to reflect the current reference.

GDG All Processing Routine

Module IEFVM4LS builds an SIOT, JFCB, and volume table entry for each GDG member, when the entire generation data group is specified by the programmer.

Patterning DSCB Processing Routine

Module IEFVM5LS completes control information in a JFCB when a new data set is to be patterned after a previously cataloged data set. The volume control block or data set pointer entry, which contains the volume serial of the volume that contains the data set, is read into main storage. Fields in the JFCB are checked for zeros, and corresponding fields from the DSCB replace them.

Error Message Processing Routine

Module IEFVMLS6 is entered and issues error messages whenever an error condition is detected by one of the JFCB Housekeeping routines. It frees the work area used by the JFCB Housekeeping routines, then passes control to the Allocate Exit routine, which sets the job-flush bit in the JCT, and passes control back to the JFCB Housekeeping Control routine.

Calculating Table Sizes

The Allocate Control routine (module IEFXCSSS) determines the sizes of certain tables that are constructed by the I/O Device Allocation routine, obtains the necessary main storage, and places the address of the block of storage reserved for each table into the allocation control block (see Figure 82). In addition, it obtains a main storage area for the IOS UCB lookup table, moves the table into the area, and modifies it if the configuration includes data cell drives. The routine adjusts the LCT to point to the expanded version of the table, and checks for and executes any VARY OFFLINE or UNLOAD commands. The routine is entered from the JFCB Housekeeping routine; exit is to the Demand Allocation routine.

Upon entry, the storage requirements for the tables needed by the I/O Device Allocation routine (except for the TIOT) are calculated. Figure 83 illustrates the calculations, and Figure 84 shows the relative positions of the tables in main storage. The required amount of main storage is calculated, and the address of the area assigned to each table is calculated. The first available byte in the main storage area is the address of the first table; the other addresses are determined by adding the size of the preceding table to its address. As each address is determined it is stored in the allocation control block.

When storage areas have been assigned to all tables except the allocate volume table (Figure 85) and the TIOT, the Queue Management Read routine is used to bring all the SIOTs for the step into the area assigned to them. The number of volumes to be used by the step is determined from information in the SIOTs, and used to calculate the size of the allocate volume table. The routine then makes a second request for main storage to accommodate the allocate volume table, and the address of the area is placed in the allocation control block.

<table>
<tr><td colspan="2">Address of Channel Load Table 4</td></tr>
<tr><td colspan="2">Address of First Empty Slot in Allocate Volume Table (AVT) 4</td></tr>
<tr><td colspan="2">Address of Potential User on Device Table (PUDT) 4</td></tr>
<tr><td colspan="2">Address of Allocate Work Table (AWT) 4</td></tr>
<tr><td colspan="2">Address of Allocate Volume Table (AVT) 4</td></tr>
<tr><td colspan="2">Address of Volume Table (VOLT) 4</td></tr>
<tr><td colspan="2">Address of Separation Strikeout Pattern 4</td></tr>
<tr><td>Number of Allocated Entries[1] 2</td><td>Number of Unallocated Entries[2] 2</td></tr>
<tr><td>Number of Bytes per AWT Entry 2</td><td>Length of AVT 2</td></tr>
<tr><td>Length (In Words) of Primary Bit Pattern 2</td><td>Number of DD Statements in Step[3] 2</td></tr>
<tr><td>Reserved 2</td><td>Number of Devices in Configuration[4] 2</td></tr>
</table>

Notes:

[1] Set to zero initially and incremented each time a data set is completely allocated.

[2] Initially set to number of data sets to be allocated (the number of DD statements in the step), and decremented each time a data set is completely allocated.

[3] The number of data sets to be allocated.

[4] The number of addresses in the I/O supervisor UCB lookup table.

Figure 82. Allocation Control Block

DD number table	$4\left\|\frac{A}{4}\right\|$
Buffer	176
Allocate	44
Channel load table (CLT)	4C
Allocate work table (AWT)	$(24 + 8\left\|\frac{D}{32}\right\|)\ A$
Potential user on device table (PUDT)	$4\ \frac{D}{4}$
Separation strikeout pattern	$\frac{D}{32}$
Each SIOT	68
Volume table (VOLT)	6S
TIOT	Determined by the TIOT Construction routine
Allocate volume table	8B
Device mask table (DMT)	$4 + (8 +\left\|\frac{D}{32}\right\|)\ F$

Legend
- $\| \|$= Next higher integer if a fraction
- A= Number of DD statements
- B= Number of volumes or devices (whichever is greater).
- C= Number of channels specified at system generation time.
- D= Number of entries in the I/O supervisor UCB lookup table.
- F= Number of entries in device mask table.
- S= Number of volume serial numbers.

Figure 83. Formulas for Determining Allocation Table Sizes

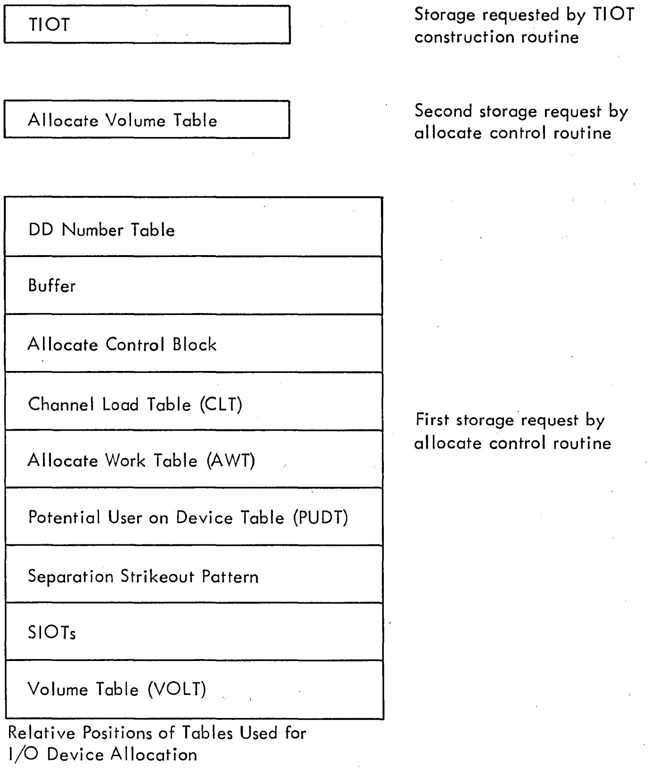

TIOT
Storage requested by TIOT construction routine
Allocate Volume Table
Second storage request by allocate control routine
DD Number Table
Buffer
Allocate Control Block
Channel Load Table (CLT)
Allocate Work Table (AWT)
Potential User on Device Table (PUDT)
Separation Strikeout Pattern
SIOTs
Volume Table (VOLT)
First storage request by allocate control routine
Relative Positions of Tables Used for I/O Device Allocation

Figure 84. Relative Positions of Tables Used for I/O Device Allocation

DD number 1	Status E 1	UCB address 2
Volume serial pointer 2		Volume affinity link 2

Figure 85. Allocate Volume Table Entry

The storage required for the TIOT is calculated in the TIOT construction routine.

Constructing Tables

The Demand Allocation routine (module IEFWA000) is entered from the Allocation Control routine when table sizes have been calculated and main storage reserved. Upon entry, the routine issues an ENQ macro instruction to prevent access to the UCBs by the Termination routine and SYSOUT writer. The first part of the routine builds the allocate work table (AWT), and, in the process, links data sets with similar device requirements (SPLIT and SUBALLOC requests). The routine then constructs the allocate volume table, and using the information previously gathered, it resolves volume affinity requests, linking entries in the allocate volume table. Finally, it calculates the device requirements of each data set, then constructs the channel load table (CLT).

Constructing the Allocate Work Table (AWT)

When the Demand Allocation routine is first entered, it constructs the allocate work table (Figure 86). The AWT contains information that describes a data set, as well as other information used for allocating units to it. One entry is constructed for each data set; most of the information is obtained as shown in Figure 87.

<table>
<tr><td colspan="2">Number of Devices Available 2</td><td>SPLIT/ SUBALLOC Link 1</td><td>Number of Devices Requested 1</td></tr>
<tr><td>Status A 1</td><td>Status B 1</td><td>Status C 1</td><td>Status D 1</td></tr>
<tr><td>Number of Volumes 1</td><td>Number of Devices Allocated 1</td><td>Number of Devices Shared 1</td><td>Number of Devices Required 1</td></tr>
<tr><td>Unit Affinity Link 1</td><td>Status 1 1</td><td colspan="2">Address of First Entry in Volume Table 2</td></tr>
<tr><td colspan="2">Possible Number of Devices in Secondary Bit Pattern 2</td><td>DD Number 1</td><td>Reserved 1</td></tr>
<tr><td colspan="4">Device Type 4</td></tr>
<tr><td colspan="4">Primary Bit Pattern Var.</td></tr>
<tr><td colspan="4">Secondary Bit Pattern Var.</td></tr>
</table>

Figure 86. The Allocate Work Table

The device type, as obtained from the SIOT, is used as the search argument in a search of the device mask table (DMT). When a matching device type is found, the bit pattern field of the device mask table is placed in the primary and secondary bit pattern fields of the AWT.

Entry	Source
Number of devices available	Device mask table
SPLIT/SUBALLOC link	SIOT
Number of devices requested	SIOT
Number of volumes	SIOT
Status A	SIOT
Status B	SIOT
Status C	SIOT
Status D	SIOT
Number of devices allocated	Inserted as devices are allocated.
Number of devices shared	Calculated
Number of devices required	Calculated
Unit affinity link	SIOT
Status 1	SIOT
Address of first entry in volume table	Calculated
Possible number of devices in secondary bit pattern	Device mask table
DD number	SIOT
Device type	SIOT
Primary bit pattern	Device mask table
Secondary bit pattern	Device mask table

Figure 87. Allocate Work Table Entry Sources

Data sets that have similar I/O device requirements are then linked together. Similar requirements are implied when the programmer specifies the following in a DD statement:

- SPLIT= indicating that two or more data sets in the same job step are to share a cylinder of a direct access device.

- SUBALLOC=stepname.ddname (or ddname) indicating that space for the data set will be suballocated from the space allocated to the data set described in the DD statement named ddname.

Pointers are placed into the SPLIT/SUBALLOC link field and the unit-affinity-link field of the AWT to link all such groups together.

The allocate volume table is then constructed. One entry is made for each volume required by the data set.

Volume Affinity Resolution

Volume affinity means that a certain volume is required for more than one data set. It may be requested explicitly by the use of the REF parameter of the VOLUME field of the DD statement, or implicitly by specifying the same volume serials in one or more DD statements. In either case, the entries in the allocate volume table are linked with pointers placed in the volume link field by the Demand Allocation routine.

Calculating Device Requirements

Information from the allocate work table is used to determine the number of devices required for the data set as follows (see Figure 88 for a parallel summary):

- For a data set marked parallel mount (the P subparameter of the UNIT field was specified in the DD statement), the number of devices required is the number of volumes on which the data set resides.

- For data sets not marked parallel mount:

  a. If both unit affinity and volume affinity apply to the data set, and if only tape devices can be used with this data set, then the number of devices requested for the data set will be allocated.

  b. If all the volumes on which the data set resides are to be shared,[1] a device will be allocated for each volume.

  c. If some of the volumes on which the data set resides will not be shared, but more devices were requested than will be shared, the number of devices requested will be allocated.

  d. If some of the volumes on which the data set resides will not be shared, but the number of volumes to be shared is at least as great as the number of devices requested, one more device will be allocated than the number of shared volumes.

Data sets marked parallel mount:
$D_1 = V_2$

Data sets not marked parallel mount:

a. If both unit affinity and volume affinity apply to the data set, and if the device type specified in the DD statement for the data set is tape only, then $D_1 = D_2$.

b. If $V_1 = V_2$, then $D_1 = V_2$

c. If $V_1 < V_2$, and $V_1 < D_2$, then $D_1 = D_2$

d. If $V_1 < V_2$, and $V_1 \geq D_2$, then $D_1 = V_1 + 1$

When: D_1 = The number of devices actually to be allocated to the data set. This number is placed in the number-of-devices-required field of the allocate work table (AWT).

D_2 = The number of devices requested for the data set.

V_1 = The number of volumes that are reserved, permanently resident, or are to be shared with at least one other data set.

V_2 = The number of volumes on which the data set resides.

The number of devices to be allocated is placed in the number-of-devices-required field of the allocate work table.

Figure 88. Calculation of Data Set Device Requirements

[1] In calculating device requirements, volumes that are reserved or permanently mounted are considered to be shared volumes.

Channel Load Assignments

For the purposes of allocation, a channel is defined as a discrete path from a device to the CPU (or main storage). The load on a channel is the number of data sets accessible through that channel. The channel load table (CLT) furnishes a place to record the channel loads. This table is built in the scheduler work area by the Allocation Control routine (module IEFXCSSS), and the allocation routines use its information about channels and their loads to manage channel and device resources efficiently.

Device allocation does not depend on physical channel addresses. Instead, the CLT defines channels by means of pointers to a list of device UCB addresses. Each pointer defines a single channel, but may point to a block of several entries. Each entry, in turn, corresponds to a single device, so that a single channel may provide access to a number of devices. This "shredding-out" of the data paths that a channel provides is illustrated in Figure 89, which also shows how more than one channel pointer from the CLT can ultimately designate a single device. The flexibility in device allocation that this scheme provides is the flexibility, for example, that the Model 2870 multiplexor channel (with its subchannels) requires.

The scheduler lookup table makes this flexibility possible by interposing one level of addressing between the CLT and the device UCBs. The Allocation Control routine builds the table, with the CLT, in the scheduler work area, constructing it in three sections of halfword entries. The first section is a copy of the device list portion of the I/O supervisor lookup table. The entries in this section contain the addresses of specific device UCBs. Addresses in the first halfword of each fullword entry in the CLT point to blocks of entries in the scheduler lookup table to define the discrete channels for the allocation routines.

For example, in Figures 89 and 90, P_1 points to the first scheduler lookup table entry for channel one. Channel one entries include all the succeeding entries to the point where the second pointer, P_2, designates the beginning of the block of entries representing devices accessed through channel two. In Figure 89, the pointers from the CLT illustrate that channel one provides a data path to the 2311, the 2314, and the 2400 devices, while channel two provides a data path to the 2321 device. Note, however, that channel two also provides a data path to the 2311 device, because the first table entry for that channel also points to the UCB for that device.

The allocation routines can keep track of all the data paths provided to a device by using an allocation channel mask. This mask is a bit configuration that subroutine IEFXDPTH builds for use by the following allocation routines:

- The device strikeout routine -- IEFX300A;
- The separation strikeout routine -- IEFXH000;
- The decision allocation routine -- IEFX5000;
- The TIOT construction routine -- IEFWCIMP.

When an allocation routine calls subroutine IEFXDPTH, it passes to it a standard parameter list that includes a pointer to a UCB and a space for the channel bit pattern used as the mask. The subroutine searches the scheduler lookup table for UCB pointers identical to the one passed, notes the channel number associated with any such pointer entry, and turns on the bit corresponding to that channel in the mask space provided by the parameter list. The subroutine then returns control to the calling allocation routine, which now has channel information at its disposal.

The second halfword of each CLT entry is the number of data sets that constitutes the load on the channel to which the first halfword points. Hence, in Figure 90, L_1 and L_2 are the respective loads on channels one and two. In the same figure, P_n points to the last channel for which there is a set of one or more scheduler lookup table entries, and L_n is the load on that channel. P points to the first field of hexadecimal Fs in the scheduler lookup table. This field separates the first section, which contains the UCB addresses, from the second section. Allocation routines use this boundary and its CLT pointer to facilitate rapid searching of the table.

The second section in the table contains sets of ten pointers each for the sub-UCBs associated with every main UCB controlling a 2321 data cell drive. Such a set exists for every 2321 device that the operating system is using. A second entry of hexadecimal Fs follows the last sub-UCB entry in the section to delimit the entries from the different type that follows. The P address in the CLT points out this quick-reference delimiter.

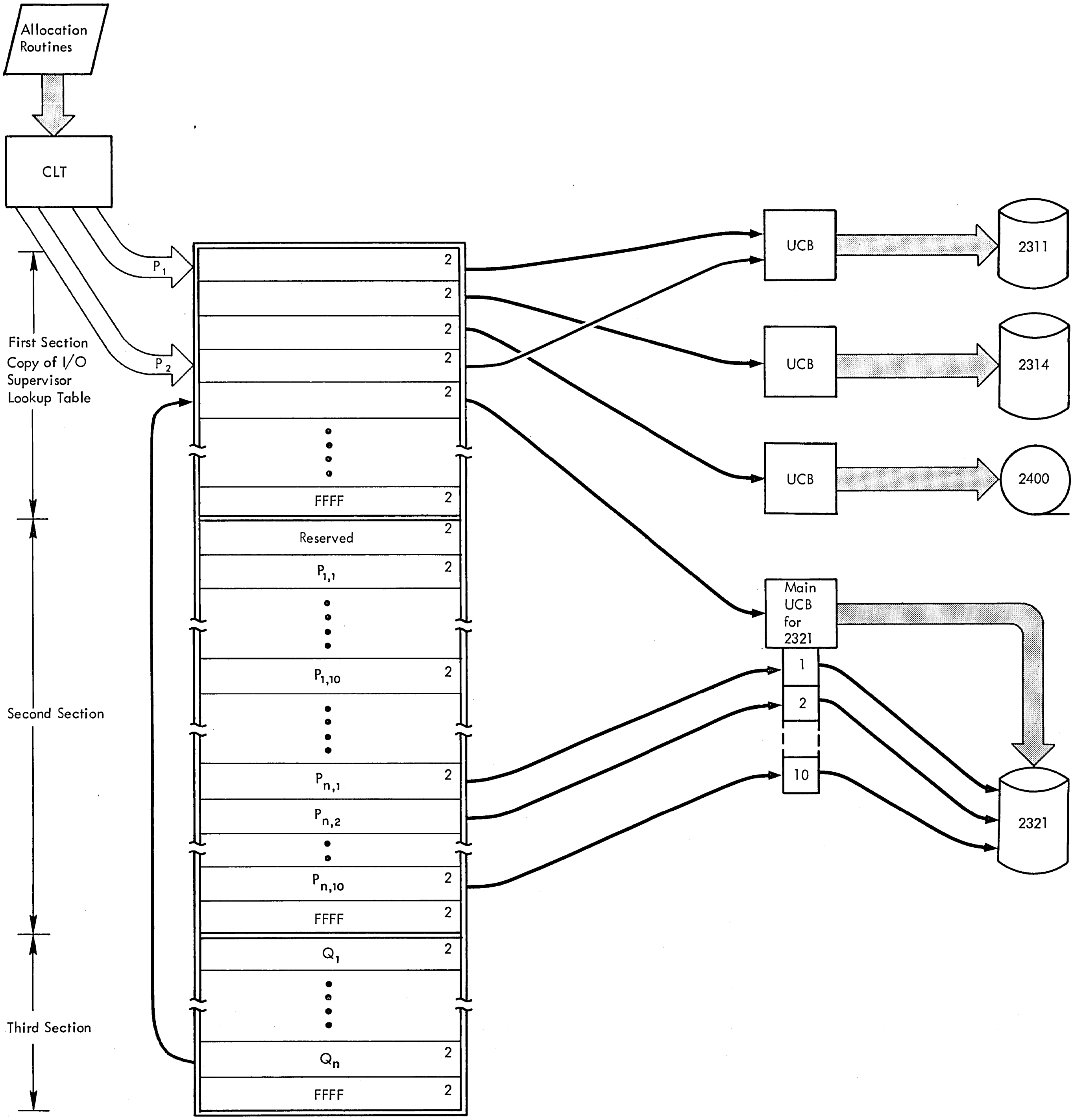


Figure 89. Scheduler Lookup Table

The third section contains pointers to the first section. These pointers relate each set of ten sub-UCBs to its 2321 device main UCB. For example, in Figure 89, Q_n is a pointer associated with the set of sub-UCB pointers $P_{n, 1}$, and it refers the set back to its proper main UCB via the pointer in the first section.

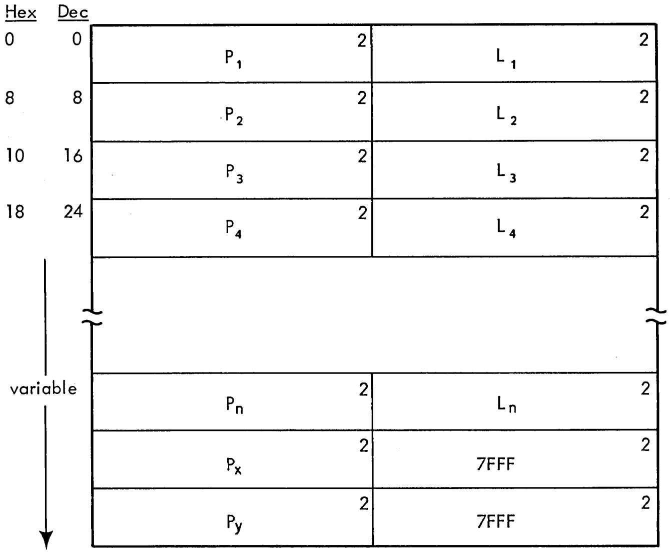


Figure 90. Channel Load Table

UNIT ASSIGNMENT

Selecting a device for assignment to a data set is essentially a process of elimination. When as many units as possible have been eliminated because of ineligibility or unsuitability, one of the remaining (equally suitable) units is assigned to the data set. Hopefully, the process of elimination will eliminate all but one unit; where a choice remains, however, the list of UCBs is scanned; when a UCB is encountered that corresponds to an eligible unit, the unit is assigned to the data set.

The first requests to be honored are those that permit no choice of units. The Demand Allocation routine processes these requests; if all requests are satisfied by demand allocation, control is passed to the TIOT Construction routine. Otherwise, the process of elimination is continued in the Decision Allocation routine. At the completion of decision allocation processing, units have been assigned to all requests except those involving public volume space. Processing of requests for public volumes is continued in the TIOT Construction and Space Request routines.

Demand Allocation

The first portion of the Demand Allocation routine (module IEFWA000), is discussed in the section, "Gathering Information"; it constructs several tables that are used during allocation. The second portion of the routine, however, begins the process of unit assignment, by performing the following functions:

- It honors explicit requests for resident direct access volumes, and reserved units.

- It reduces the range of eligible devices.

- It honors explicit and implied requests for specific units. When these functions have been performed, it passes control to the Decision Allocation routine (if there are outstanding requests), or to the TIOT Construction routine (if all requests have been honored).

Allocation of Resident and Reserved Volumes

When the allocate work table and the allocate volume table have been built, requests, by volume serial, for resident and reserved direct access volumes or allocated units (including requests for the volume containing a system input data set) are processed.

The Demand Allocation routine obtains the last entry in the device mask table (this entry has all bits corresponding to direct access devices set on), and examines the list of UCBs, looking for a match to the volume serial requested. When a match is found, the UCB is inspected to insure that the volume is resident, allocated, or reserved, and the data set's AWT entry is inspected to insure that the the device is eligible for allocation to that data set. Finally, the Device Strikeout routine is used (as a subroutine) to perform the allocation. If an error is detected, control is passed to the Allocation Recovery routine; otherwise, device range reduction is performed.

Device Range Reduction

The primary bit pattern of a data set's AWT entry points to every unit of the device types that could be used by the data set. Some of these units, however, may be ineligible for allocation to the data set because their status or volumes mounted may be inconsistent with the request. Device range reduction is the process of eliminating from consideration those units that are ineligible for allocation in response to the requests made in the step.

The following are ineligible for allocation in response to some requests or all requests:

- Units whose status is off-line, or is being changed to off-line, are never eligible for allocation unless they are marked as being communication devices.

- The primary console input and output devices are never eligible for allocation.

- Units containing volumes that are both reserved and private are not eligible for allocation unless the correct volume serial was specified.

- Units containing system residence volumes are ineligible for allocation in response to private volume requests unless the correct volume serial is specified.

- Units that are being tested by the Online Test Executive Program (OLTEP) are never eligible for allocation. The Demand Allocation routine determines whether a unit is being tested by OLTEP by checking bit 5 in byte 1 of the UCB.

- Allocated units are ineligible, except for the following:

  a. A demountable, direct access device containing a public volume, unless the data set is public.
  b. Resident direct access devices.
  c. A demountable, shareable, direct access device that contains a volume matching the volume requested in a shareable request.
  d. Communication devices.

The routine processes each allocate work table entry in turn. When a unit is found to be ineligible for allocation to the data set, the corresponding bit in the primary bit pattern of the data set's AWT entry is set to zero, and the number of available devices is reduced by one. If the number of available devices becomes less than the number of devices required, control is passed to the Allocation Recovery routine. Otherwise, the next data set request is processed.

Allocation of Specifically Requested Units

The Demand Allocation routine next honors requests for specific units, and requests that can be satisfied only by specific units because of device range reduction and previous allocation. If, when all such requests have been honored, there are no unsatisfied requests for devices, the Demand Allocation routine passes control to the TIOT Construction routine.

If, however, there are requests for which no units have been allocated, the routine passes control to the Automatic Volume Recognition (AVR) routine (if it has been included in the system). If AVR has not been included in the system, the Demand Allocation routine passes control to the Decision Allocation routine.

Automatic Volume Recognition

If Automatic Volume Recognition (AVR) is specified when the system is generated, the operator can mount volumes required for subsequent job steps as soon as units become available. The AVR routine (modules IEFXV001, IEFXV002, and IEFXV003) recognizes the volumes that have been mounted for the step being initiated, thus saving the time that the system would otherwise spend waiting for the operator to find and mount them. Module IEFXV001, the first of the three AVR modules to receive control, uses four subroutines contained in module IEFXV002 as follows:

- READSER - for reading a volume label into main storage, for extracting the volume serial number and for placing it into the UCB for the corresponding device.

- SERBUILD - for building a list of volume serial numbers from requests for unmounted volumes that AVR is to handle.

- PATTEST - for verifying that a specified device is acceptable to all requests in the unit and volume affinity chains.

- PUTLIST - for issuing mount messages for all unmounted volumes that AVR routines are to process.

The third AVR module, IEFXV003, processes unmounted tape requests. In doing this, it determines the most satisfactory allocation solution to tape requests (based on device configuration) for 9-track tape volumes on either type 3400 or type 2400 tape drives. Module IEFXV001 does the rest of the processing for AVR requests.

The AVR routine allocates units to requests for 2311, 2305, 2314, and 3330 direct access volumes, premounted 7-track tape volumes having the tape density specified when the system is generated, and 9-track tape volumes. The allocation request must specify volumes by volume serial or by a data set name that implies a volume serial.

The AVR routine first allocates the units containing volumes required for the current job step. When this has been done, the routine attempts to satisfy any remaining requests for 2311, 2305, 3330, and 2314 direct access volumes and for 9-track tape volumes. It allocates units with no volumes mounted, then (if more units are needed) it unloads units containing volumes not required for the current step. Unless enough units can be made available, and unless all required volumes can be mounted, the system terminates the job.

Processing Requests for Mounted Volumes

The AVR routine first satisfies requests for volumes that are already mounted. It searches for these volumes by examining all UCBs corresponding to 2311, 2305, 3330, and 2314 direct access units, 7-track tape volumes having the tape density specified when the system was generated, and 9-track tape units. If such a unit is online and in the ready state, but the serial number field in the UCB is zero, it means that the volume was mounted since the start of the previous job step and the label has not yet been read.

In this case, the READSER subroutine in the second module (IEFXV002) reads the volume label into main storage and extracts the volume serial number from the volume label as follows:

- If the label is an IBM standard tape label, the AVR routine records the volume serial number in the UCB and sets a bit in the UCB to indicate an IBM standard label.

- If the label is an American National Standard label, the volume serial number must be translated from American National Standard Code for Information Interchange (ASCII) to EBCDIC. The AVR routine checks a bit in the CVT to determine whether the ASCII Translation routine (SVC 103) was included in the system at system generation time. If so, the AVR routine sets a bit in the appropriate UCB to indicate an American National Standard label and passes control via SVC 103 to the ASCII Translation routine, which converts the volume serial number to EBCDIC. Then the AVR routine records the EBCDIC number in the appropriate UCB and sets a bit to indicate an American National Standard label. If the ASCII Translation routine was not included, the AVR routine sets a return code to indicate an error condition exists.

- If the label is a nonstandard label, (i.e., not IBM standard or American National Standard) the AVR routine passes control either to the IBM-supplied subroutine IEFXVNSL to indicate an error condition or to a user-supplied IEFXVNSL that processes the nonstandard label. If IEFXVNSL is supplied by the user, the volume serial number pointed to by the user routine on the return to AVR is recorded by the AVR routine in the UCB, but no bit is set.

- If the tape has no label, the AVR routine issues a request to the operator to unload the tape.

As each volume serial is recorded, the AVR routine determines which volumes are needed for the current job step. It searches the VOLT for a match to each volume serial; when a match is found, the volume has been specifically requested for the current job step, and the AVR routine passes control to the Device Strikeout routine, which performs the actual allocation to satisfy all requests for the volume and unit. The AVR routine uses the External Action routine to notify the operator and unload the volume if any of the following occur:

- The unit has been allocated to a different volume.

- The Demand Allocation routine has allocated the volume to a different unit.

- The volume is a tape volume requested for a 3400 series tape drive but already mounted on a 2400 series tape drive.

Processing Requests for Unmounted Volumes

Before the AVR routine requests the operator to mount additional volumes, it determines whether there are enough units available to contain them:

1. It counts the units that are online, but not allocated by type (2311, 2314, 2400, 2400-3, 3330, etc.) and by state (ready or not-ready).

2. It compares the number of units of each type that are required for the step, with the corresponding count of units in a not-ready state. Units in a not-ready state do not have volumes mounted.

   For tape requests, the AVR routine attempts to satisfy requests for a specified unit type by using the counts of other compatible unit types as well as using the count of corresponding units that are in the not-ready state. As an example, a request for a 2400 tape drive with a density of 800 BPI could be satisfied by using either a type 2400-4 (dual density) tape drive or a 3400-4 (dual density) tape drive.

3. If there are not enough units in a not-ready state available, the routine determines whether there are enough unallocated units in a ready state to satisfy the outstanding requests. Such units have volumes mounted; the volumes may be retained volumes, or they may contain data sets, and if so, the routine will not use them unless it is necessary to do so.

If the AVR routine has not found enough online, unallocated units to satisfy all requests, it passes control to the Allocation Error Recovery routine to issue a list of offline units that can be made available. The operator may reply with three-character unit names to place the units in online status or he may cancel the job.

If the AVR routine has found enough units, it furnishes the operator with mounting messages that list the volumes to be mounted and unit types required. If the DISPLAY DSNAME command is in effect, the mount messages include the names of the first nontemporary data sets on the volumes to be mounted. (See "The DISPLAY DSNAME Command.") The External Action routine unloads any units in the ready state that must be used.

When the mounting instructions have been issued, the AVR routine obtains main storage from the system queue area (SQA) for a mount verification communication area (MVCA), constructs a dummy MVCA extension (MVCAX), places the MVCA in the IEFVPOST ECB list, then waits for a device-end interruption to signal that a volume has been mounted. When the interruption occurs, the Unsolicited Device End Posting routine (module IEFVPOST) posts the appropriate ECB. Then the AVR routine uses the READSER subroutine to copy the volume serial number from the label of the volume that has been mounted as described in the section <u>Processing Requests for Mounted Volumes</u>.

After the volume serial number is in the UCB, the AVR routine checks whether the volume is being waited on by AVR for the job step and, if it is, allocates it by giving control to the Device Strikeout routine.

If the volume that was just made ready (i.e., that was just mounted) is needed by the job step but is not one for which the AVR routine was waiting, either IEFXV003 (for tape devices or module IEFXV001 (for direct access devices) determines the availability of other eligible devices in the not-ready state on which the operator may mount volumes to satisfy the requests.

If eligible devices in the not-ready state are unavailable, the module checks the availability of eligible on-line devices that can be unloaded. If such devices are available, the AVR routine unloads them. If unloadable devices are not available, the module gives control to the allocation recovery routines.

If the volume that was just made ready is not needed by the job step, an insufficient number of acceptable not-ready devices may remain for satisfying the current requests. When this happens, the AVR routine unloads the newly-mounted, unneeded volume and issues a diagnostic message.

When all units have been allocated, the AVR routine removes the Mount Verification Communication Area from the ECB list and frees the main storage occupied by the MVCA. The routine then passes control to the Decision Allocation routine (if there are still data sets to which no unit has been allocated) or to the TIOT Construction routine (if all volume requests have been honored).

<u>Decision Allocation</u>

The Decision Allocation routine (module IEFX5000) allocates units to all unallocated data sets requiring private volumes or specifying volume serials, to data sets passed by a previous step or requiring retained volumes (if the volumes are mounted), and to any other data sets whose eligible units are reduced (by the decision allocation process) to the point where a choice no longer exists.

There are three steps in the decision allocation process:

- <u>Unit Elimination</u>: channel and unit separation requests are processed to reduce the number of units eligible for allocation to a data set.
- <u>Data Set Selection</u>: the data set whose requirements are most restrictive is chosen.
- <u>Unit Selection</u>: the most suitable of the eligible units is chosen and allocated to the data set.

When unit elimination has been performed for all data sets, the data set selection and unit selection steps are repeated for each data set until all allocation requests have been honored. If an error is encountered, or if there are allocation requests that cannot be satisfied by the available units, the Allocation Recovery routine is executed. When unit assignment is complete, control is passed to the TIOT Construction routine.

Unit Elimination

If an allocation request includes channel or unit separation from another data set to which a unit has been allocated, the units whose allocation would violate that request are declared ineligible (eliminated). The corresponding bits in the primary bit pattern of the data set's AWT entry are turned off.

The Decision Allocation routine scans the AWT for unit and channel separation requests, then uses the Separation Strikeout routine to eliminate the ineligible units. When all such requests have been processed, the number of data sets assigned to each channel is added to the channel load table (CLT), which is later used to balance the channel workload.

Passed Data Sets

In the receiving step, units are allocated to passed data sets on the basis of a UCB scan; when a UCB is found that shows a volume of the data set to be mounted on the associated unit, the unit is allocated. Devices with volumes containing passed data sets, and volumes retained for another job, are not generally allocated to data sets that have not been passed.

Data Set Selection

When the number of eligible devices has been reduced as much as possible, (by device range reduction, previous allocation, and unit elimination), the Decision Allocation routine performs the two-step process of selecting a data set and allocating a unit.

Data sets are selected in order of restrictiveness of request (the most restrictive request can be satisfied by the fewest devices). The most restrictive request is selected first in order to avoid allocating all of its eligible devices to other data sets. Figure 91 shows a situation in which two data sets require one unit each:

	Unit 1	Unit 2
Data Set A	X	X
Data Set B	X	

Figure 91. Data Set Selection Example

Data set A can use either unit, but data set B can use only unit 1. Therefore, if data set A were selected first, it might have unit 1 allocated to it, and no unit would be available for data set B.

If two or more data sets have the same number of eligible units, the relative restrictiveness of the requirements is determined based upon the presence or absence of certain attributes. Each data set's AWT entry is scanned, and a weighting factor added for each uniqueness attribute found to be present. The data set totals are compared, and the data set with the highest total is considered the most restrictive. Figure 92 lists the uniqueness attributes, and the weighting factor used with each.

Attribute	Factor
Received data set (passed by previous step)	8
Channel separation requests	4
Unit affinity request	2
Channel affinity request	1

Figure 92. Data Set Uniqueness Attributes

If two or more data sets have equal totals, they are selected in the order in which their DD statements appeared in the input stream.

Unit Selection

When a data set has been selected, a unit is selected and allocated to it. Units are considered in the following order:

1. If there are eligible devices on more than one channel, the channel with the greatest number of free units of the type requested is chosen.
2. If more than one channel has the same number of free units of the requested type, the channel with the lightest load is chosen; the device on that channel with the fewest potential users is chosen.
3. If there are several units with the same number of potential users, the first such device in the I/O supervisor UCB lookup table is chosen.

When a unit has been chosen for allocation to a data set, the Device Strikeout routine is used to complete the allocation. When units have been allocated to all requests except non-specific public volume requests, control is passed to the TIOT Construction routine.

TIOT CONSTRUCTION

The task input/output table (TIOT) is built by the TIOT Construction routine (module IEFWCIMP), which is entered from the Demand Allocation or Decision Allocation routine. The TIOT Construction routine calculates the amount of storage required to build the TIOT, and constructs the entries.

Storage Requirement Calculation

Before the TIOT is built, the TIOT Construction routine calculates the amount of storage that will be required (see the first formula in Figure 93). When there are non-specific public volume requests, the routine builds a list of units eligible for allocation to each public volume request, and places the list in the data set's TIOT entry. The TIOT must therefore be large enough to accommodate the extra information. When, in the Space Request routine, the required units are finally allocated, these entries are compressed to the size indicated by the second formula in Figure 93.

Space required to build TIOT = $28+16N_1+4N_2+4(N_3N\ )+8N\ +N$

Space required for compressed TIOT = $28+16N_1+4N_2$

Where:

N_1 = Number of DD statements in the step.

N_2 = Number of devices allocated to the step.

N_3 = Number of non-specific requests for space on public volumes.

N = Number of devices available for public volumes.

N = Number of entries in I/O Supervisor UCB list table.

Note: The TIOT is built by the TIOT Construction routine, then compressed by the Space Request routine.

Figure 93. Formulas for Finding TIOT Storage Requirements

TIOT Entry Construction

When the storage requirement has been calculated, and the required amount of storage has been obtained, a TIOT entry is constructed for each data set associated with the step. If a job library has been specified, if a step library has been specified, or if the current step is to be executed by a program created in a previous step, entries are also created for the joblib, steplib, and fetch data sets. The entries are processed in input stream sequence using the information sources shown in Figure 94.

As the entries are built, any logical links between data sets are recorded. A data set requesting volume affinity to another data set causes the other data set to be identified as a primary volume affinity data set. The requesting data set is identified as a secondary volume affinity data set; it is linked (via TIOT entry number) with the primary. Similar identification and chaining is performed when suballocation, unit affinity, and split cylinder requests are encountered. This technique permits control to be maintained so that only one mount message is issued (by the External Action routine) even though many data sets may reside on the same (currently unmounted) volume.

Entry	Source
Jobname	JCT
Stepname	SCT
Stepname of step in which procedure was requested	SCT
Length of entry	Calculated
Status A	Calculated
ddname	SIOT
Address of JFCB	SIOT
Status C	Calculated
Status B	Calculated
Address of UCB	I/O Supervisor UCB lookup table
Slot for UCB	I/O supervisor UCB Lookup Table

Figure 94. Task Input/Output Table Entry Sources

Public Volume Processing

When the TIOT Construction routine is entered, units have been allocated to all data sets except those that require space on public volumes, and for which there is a choice of eligible units. When it creates the TIOT entry for such a data set, the TIOT Construction routine builds (in the entry) a list of eligible units.

The first unit in the list is the unit that appears to be the most suitable unit with which to satisfy the request. It is chosen as follows:

1. Of the channels that contain eligible units, the one that has the lightest data set load is chosen.
2. The unit selected as the most suitable is the direct access device (on that channel), with the smallest number of users, that is not reserved, does not contain a private volume, and whose volume does not contain any passed data sets.

It is assumed that space will be available on the most suitable unit and that it can therefore be allocated to the data set. The Separation Strikeout routine is entered to process requests for channel and unit separation from this data set, then the other eligible units are placed in the list in descending order of suitability:

1. Other direct access devices on the system with the characteristics of the first unit.
2. Direct access devices containing volumes that are not private or reserved but contain passed data sets.
3. Direct access devices with demountable volumes that are currently allocated to other jobs as private or nonshareable volumes are added to the list. They are marked as such to prevent allocation until the unit becomes unallocated.
4. Direct access devices with demountable volumes, that are currently allocated to other jobs, are added to the list. They are marked as such to prevent allocation until the unit becomes unallocated.
5. Direct access devices (on other channels), that have volumes containing passed data sets, are added to the list.

When the list (and the TIOT entry) has been completed for a data set, the process is repeated for succeeding public volume requests. When processing is complete, control is passed to the External Action routine for volume mounting.

SPACE ASSIGNMENT

When the required units have been allocated to all data sets requesting private or specific volumes, and a list of available units has been made for those data sets requesting public direct access space, the function of obtaining and assigning direct access space must be performed. The function is performed by the following routines.

- The External Action routine (module IEFWD000), issues mounting instructions for volumes and verifies that the correct data cell volumes and direct access volumes requiring space allocation for new data sets are mounted.
- The Space Request routine (module IEFXT00D), which requests space from the Direct Access Device Storage Management (DADSM) routines, and completes the allocation of units to data sets requiring public volume space.
- The VARY Allocation Interface and TIOT Compression routine (module IEFXT002), which changes the status of a device from online status (allocated or unallocated) to console status, compresses the TIOT, places scratch volume information in the JFCB, and, if MSGLEVEL=(X,1), places allocation messages in SMBs. If the routine is entered from the Space Request routine after it has detected an unrecoverable error, the TIOT Compression routine compresses the TIOT, then passes control to the Allocation Nonrecovery Error routine.
- The Extended External Action routine (module IEFWEXTA), which verifies that the correct direct access volumes containing old data sets have been mounted.
- The DADSM Error Recovery routine (module IEFXT003), which determines what action should be taken when a request for space on a direct access volume is not honored.

Mounting Volumes

The External Action routine (module IEFWD000) is entered from the TIOT construction module to issue mounting instructions to the operator, to verify that volumes requiring space allocation for new data sets and all requested data cells have been mounted, and to unload any devices known to contain the wrong volumes. When processing is complete, control is passed to the Space Request routine.

When the routine is first entered, it makes a check of the volumes recorded in the UCBs to determine whether any volumes required by the current step are mounted on the wrong units. If so, and if SMF volume accounting has been specified for the system, the External Action routine issues an SVC 78 (LSPACE) for each direct access volume that is incorrectly mounted. The SVC 78 routines, which are described in the Direct Access Device Space Management PLM, create a type 19 SMF record and cause it to be placed in the SMF data set.

On the return from the SVC 78 routine, the External Action routine requests the operator to demount and retain the volume. In the case of tape volumes, no SVC 78 is issued; the External Action routine requests the operator to demount the volume, and unloads it.

After the check of unallocated devices has been made, the routine scans the TIOT for unload requests. These requests occur when a volume required by the job step cannot be mounted until the currently mounted volume is removed from the unit. The routine issues SVC 78 for such volumes if they are direct access volumes, or unloads them if they are tape volumes, then requests the operator to demount them.

When the unload requests have been satisfied, the External Action routine scans the TIOT for mount requests, issues mounting instructions to the operator, and marks the appropriate TIOT entries so that verification can be done when the unit becomes ready. If the MONITOR DSNAME command is in effect, the mount messages to the console or TSO terminal operator include the names of the first eligible nontemporary data sets associated with the TIOT entries for which mount messages are to be issued. (See "The MONITOR DSNAME Command.")

While issuing mounting instructions for a volume, the External Action routine checks the TIOT to determine whether the volume contains an old data set. If so, verification of the volume is delayed until the Extended External Action routine (module IEFWEXTA) is executed at the end of allocation. (Data cell volumes are an exception; all requested data cell volumes are verified by the External Action routine along with the volumes requiring space allocation for new data sets.) The External Action routine indicates that verification of a volume is delayed by placing the address of the mount verification communication area (MVCA) for the volume in the allocation/termination communication area (ATCA). Then the routine places the correct volume serial number in the appropriate UCB and sets the mount pending bit on in the data management field of the UCB. If MCS is in the system, the routine places the DOM ID of the volume's mount message in the DOM ID field of the mount verification communication area extension (MVCAX) for the volume.

After issuing the mounting instructions, the routine issues a WAIT macro instruction while the direct access devices for volumes that require space allocation and for data cell volumes become ready. In order to allow the Termination routine, the SYSOUT writer, and allocation for another task access to the UCBs during the wait period, the routine issues two DEQ macro instructions for the UCBs. However, when the volumes are finally mounted, the routine issues two ENQ macro instructions to prevent further access to the UCBs by the Termination routine, by the SYSOUT writer, and by allocation routines for another task. When the devices are ready, the routine checks the volumes that have been mounted. If labeled direct access volumes are specified, the volume labels are read in and verified; if a direct access volume is requested but no specific volume is specified, the volume is considered verified when the device becomes ready. Tape volumes are checked in the Open routine.

If the routine must wait for a device end, it obtains main storage for the mount verification communication area (MVCA) and the MVCAX associated with the requested volume. Then it places the MVCA in the IEFVPOST ECB chain. When the device-end interruption occurs, IEFVPOST obtains the device-end ECB from the MVCAX, the system post routine posts it, and the External Action routine verifies the volume. Then the routine removes the MVCA and the MVCAX from the ECB chain, but keeps them intact for the Extended External Action routine (module IEFWEXTA).

If an incorrect volume is mounted, the operator is again issued a mounting message containing the correct volume identification. When all direct access devices that have become ready have been checked, the External Action routine issues a WAIT macro instruction (if there are additional direct access volumes to verify); the ECB is posted by the resident IEFVPOST routine when a device whose volume is to be verified becomes ready.

When all required mount and remove messages have been issued, and all required direct access volumes have been correctly mounted, the External Action routine passes control to the Space Request routine for public volume allocation.

Obtaining Space

The Space Request routine (module IEFXT00D) is entered from the External Action routine. At this point in the I/O device allocation processing, units have been allocated to all data sets except those requiring space on public volumes. In such cases, an ordered list has been created of eligible units; on the assumption that the first choice units can be allocated to those data sets, all volumes known to be initially required have been mounted.

The Space Request routine attempts to obtain direct access space for new data sets from DADSM. When such space is not available, it attempts to find space on another volume (it uses the list of eligible volumes provided for data sets requiring space on public volumes); if it cannot obtain space on currently available devices, it passes control to the Allocation Recovery routine. When all requests have been satisfied, it compresses the TIOT entries for data sets requiring public volume space, and passes control to the Extended External Action routine.

When it is entered, the Space Request routine scans the TIOT for new data sets requesting space on direct access volumes, and processes each in turn. Volume affinity, suballocation, and split cylinder secondaries are processed immediately after their primaries.

The appropriate SIOT and JFCBs are read into main storage (if SUBALLOC or SPLIT is specified, two or more JFCBs are required), and SVC 32 is issued to obtain the space. If the return is a normal return, the JFCB of the requesting data set is updated with the identification of the volume on which space was allocated, and the TIOT is scanned for the next request.

When an error return occurs, the Space Request passes control to the DADSM Error Recovery routine (module IEFXT003), which attempts to recover by allocating space on another volume or another unit. If, however, the request was for absolute tracks or suballocation, or if volume serials were specified and all specified volumes have been tried, no recovery is possible. In these cases control is passed to the Allocation Recovery routine, and the step is canceled; in the other cases, recovery is attempted as described in the section "Allocation Recovery."

Compressing the TIOT

When all requests have been processed, the Space Request routine passes control to the TIOT Compression routine (module IEFXT002), which compresses the TIOT to its final size and performs terminal functions. Each TIOT entry that requests space on public volumes is reordered so that the UCB addresses of the units actually allocated are first in the list of UCB pointers. The valid portions of the TIOT are then shifted over the unused UCB information; the amount of storage required for the completed TIOT is calculated (see the second formula in Figure 93), and the excess storage is released.

When the TIOT is complete, the TIOT Compression routine tests the JCT for the allocation message level of the job. If the allocation message level is one (MSGLEVEL = (x,1)), the allocation messages for the job are written into SMBs. If the allocation message level is zero, no messages are written.

Volume information is then added to the JFCBs for data sets that requested scratch volumes. Messages specifying the units allocated are placed in an SMB (obtained by the Allocate Entry routine), and the SMB is placed in the output queue.

In addition, this routine provides an interface with the VARY command when a device is to be changed from an online status to a console status. A UCB search is performed to see if the change status bit and the primary console bit in the UCB are on. Whenever both bits are on, the device is to be made an operator console. If the device is allocated to a job step, control is passed to the Allocation Exit routine. The next time the Vary Allocation Interface and TIOT Compression routine is entered, it performs the same processing. If the device is no longer allocated to the job step, it turns off the change status bit, thus indicating that the device is a console, and passes control to the Extended External Action routine.

Verifying Volumes That Do Not Require Space Allocation

The Extended External Action routine (module IEFWEXTA) is entered from the TIOT Compression routine to verify volumes that do not require space allocation, i.e. volumes that contain old data sets. (Volumes that require space allocation for new data sets and all data cell volumes are verified in the External Action routine.)

When the routine is first entered, it determines whether there is a volume that requires verification: it checks whether the External Action routine has placed the address of an MVCA in the allocation/termination communication area (ATCA). If no address exists, then no volume needs verification, so the routine passes control to the Allocation Exit routine. However, if there is an address in the ATCA, then there is a volume that requires verification, so the routine issues two DEQ macro instructions to allow access to the UCBs by the Termination routine and SYSOUT writer and by the Allocation routine so that allocation can be started for another task. Then the routine scans the TIOT for a UCB with the pending bit on in the data management field.

The routine determines from the UCB whether the device has become ready. If so, the routine verifies the volume. If an incorrect volume is mounted, the routine issues a demount and a mount message to the operator to correct the error. If MCS is in the system, the routine issues a DOM macro instruction to delete the mount message for the volume.

If the routine must wait for a device end in order to have a volume mounted, it places the MVCA in the IEFVPOST ECB list, issues a DEQ macro instruction to allow the Allocation routine to continue allocation for another task, and waits for the Unsolicited Device End Posting routine (module IEFVPOST) to post the ECB after the device end interruption. When the ECB is posted, the Extended External Action routine issues an ENQ macro instruction and determines whether the job has been canceled. If so, it passes control to the Allocation Nonrecovery routine. If the job was not canceled, the routine determines whether there are more volumes to be verified. If there are, the routine issues a DEQ macro instruction to release the UCBs and waits for the additional device end interruptions.

After all volumes have been verified, the routine passes control to the Allocation Exit routine.

COMMON SUBROUTINES

The processing involved in recovering from allocation errors, in allocating selected units to a data set, in eliminating ineligible units from further consideration, and in posting unsolicited device interruptions needs to be performed at several points in the I/O Device Allocation routine. This processing is performed by the following routines:

- The Allocation Recovery routines, which are executed whenever a lack of available devices or a lack of available direct access space is discovered. The routines allow the operator to specify which of several alternative recovery procedures is to be executed, then execute the selected procedure. The recovery procedure may include waiting for another task to terminate (so that the I/O devices used in its execution become available), or it may involve ignoring separation requests, or using a unit made available by operator action.
- The Device Strikeout routine, which is executed when a unit has been selected for allocation to a data set. It modifies the primary and secondary bit patterns in the AWT entries to complete the allocation of that data set (and of data sets linked to it by unit affinity, volume affinity, suballocate requests, or split cylinder requests), and makes the unit unavailable to other data sets.
- The Separation Strikeout routine, which makes units unavailable to a data set if those units would violate its channel and unit separation requests.
- The Unsolicited Device End Posting routine, which posts the ECB when a device with a volume to be mounted and verified becomes ready.

Allocation Recovery

The Allocation Recovery routines are entered when a lack of available devices (or volumes), or a lack of available direct access space, makes it impossible to complete allocation for a job step. The routines are:

- Allocation Recovery routine (module IEFXJIMP)
- Nonrecovery Error routine (module IEFXKIMP)
- Allocation-Wait-for-Unallocation routine (module IEFSD195)
- Allocation message module (module IEFSD096)
- Allocation-Wait-for-Space-Decision routine (module IEFSD097)

Lack of Available Devices

When a lack of available devices (or volumes) is discovered after the TIOT has been constructed, control is passed to the Nonrecovery Error routine (module IEFXKIMP). The routine loads an error code into the LCT and passes control to the Allocation Exit routine which loads an error code into register 15, thus causing the step to be canceled.

If, however, a lack of available devices is discovered before the TIOT has been constructed, that is, in the Demand Allocation routine or Decision Allocation routine, an allocation recovery is attempted. The Allocation Recovery routine (module IEFXJIMP) is entered and issues a message to the operator informing him of the situation. The message provides the operator with a list of units that are eligible but not available for allocation because of their status. Such units include those whose status is off-line (exception off-line unit being tested by OLTEP), and units that are allocated to a reading or writing task.

In addition to the list of units, the message indicates the recovery procedures that may be performed. The operator receiving the message uses the REPLY command to specify the procedure to be performed. The operands available to the operator, and the results of their use, are described below.

CANCEL: If no recovery is desired, the operator may use the single operand "CANCEL". The Allocation Recovery routine loads an error code into the LCT and passes control to the Allocate Exit routine, which initiates flush mode processing.

Unit Name: The operator may reply with a three-character unit name. If the named unit is off-line, its status is changed to on-line, and control is passed to the Allocate Control routine. The allocation process is then repeated, using the additional device.

NOSEP: If channel or unit separation was requested for a data set, the operator may use "NOSEP" in his reply. The Allocation Recovery routine then sets a bit in the LCT to indicate that separation requests are to be ignored, and passes control to the Allocate Control routine so that the allocation process can be repeated.

WAIT: The operator may include the operand "WAIT" in his reply, thus causing the routine to wait for a job step or system task to terminate (and release the I/O devices that were allocated to it). While the routine is waiting, the operator may issue a STOP RDR or STOP WTR command to terminate a specific system task, and make its unit available to the current job step. He may issue a CANCEL command at any time, to cancel the job.

When the WAIT operand is used control is passed to the Allocation-Wait-for-Unallocation routine (module IEFSD195). This routine issues a DEQ macro instruction that allows termination routines (performing other system tasks) access to the UCBs. The routine also issues a DEQ macro instruction to allow access to the UCBs by the Extended External Action routine, which may be verifying volumes for another task. The routine then issues a WAIT macro instruction, specifying the ECBs in the allocate/IEFVPOST communication block and in the command scheduling control block (CSCB).

After either ECB has been posted, the Allocation-Wait-for-Unallocation routine regains control, issues two ENQ macro instructions (to prevent access to the UCBs), and determines which ECB has been posted.

If the ECB in the CSCB was posted, the POST was issued by the Command Scheduling routine in response to a CANCEL command, indicating that the operator has canceled the job. An error return code is loaded into the LCT before control is passed to the Allocate Control routine, so that the allocation process will be performed in the flush mode.

If the ECB in the allocate/IEFVPOST communication block was posted, the POST was issued by the Terminate routine after a job step or system task had terminated, releasing the I/O devices that had been allocated to it. Control is passed to the Allocate Control routine, and the allocation process is repeated, using the additional devices obtained from the terminated job step or system task.

Lack of Available Space

When the Space Request routine encounters an error return from the DADSM Allocation routine (SVC 32), it determines whether the request included absolute tracks, suballocation, or split cylinders. If so, control is passed to the Nonrecovery Error routine (module IEFXKIMP), which causes the job to be canceled. In the other cases, however, a recovery procedure is initiated.

The first step in the recovery procedure is to try to mount a new volume on the allocated unit. The Space Request routine passes control to the External Action routine, which requests the operator to remove the currently mounted volume and to mount a scratch volume. When the scratch volume has been mounted, the request for space (SVC 32) is repeated.

Remounting is not possible, however, if the unit has one or more of the following attributes:

- The volume is not physically demountable.

- The unit is system residence.

- The unit is reserved.

- The currently mounted volume contains data sets allocated to another job.

- The unit will not accommodate the requested record size.

If remounting is not possible, and the request is for a private volume, control is passed to the Nonrecovery Error routine and the step is canceled. If the data set requires space on a public volume, however, the Space Request routine attempts to obtain space on one of the other units in the list of eligible units in the TIOT entry. The list of eligible units may contain a unit with a pending mount request for a volume that does not require space allocation, i.e., a volume that contains an old data set. Therefore, the verification of this volume is delayed until the Extended External Action routine is executed at the end of allocation processing, and the mount remains pending until the verification is completed. If the Space Request routine encounters a unit with a mount pending, it passes control to the External Action routine requesting an immediate mount of the volume.

When it has unsuccessfully requested space on all units in the list that are not separation violators, the Space Request routine passes control to the Allocation-Wait-for-Space-Decision routine.

The Allocation-Wait-for-Space-Decision routine (module IEFSD097) determines whether it is feasible to wait for termination to occur before further recovery attempts are made. It is feasible to wait only if:

- The request is for public space on any one of a number of units, or on a specific, non-private unit.

- There is at least one user (in addition to the current job step) on any one of the units from which space was requested.

If both conditions are true, a termination may release space occupied by a currently active data set. If either condition is not met, control is passed to the Space Request routine, which requests space on the remaining units in the list in the TIOT entry.

If both conditions are true, the Allocation-Wait-for-Space-Decision routine passes control to the Wait-for-Unallocation routine. This routine determines whether the request includes unit or channel separation.

If channel or unit separation was requested, the Wait-for-Unallocation routine issues a message to the requesting operator. The message specifies the possible options: canceling the job, ignoring separation requests, or waiting for a termination to occur. The operator indicates his choice by issuing a REPLY command, with one of the operands described below.

CANCEL: The operator uses this operand if he wishes to cancel the job. A return code is loaded, and control is passed to the Allocate Exit routine.

NOSEP: If the operator uses this operand, control is passed to the Space Request routine, which attempts to obtain space on units that violate separation requests. If the attempt is unsuccessful, control is passed back to the Wait-for-Space-Decision routine, which determines whether a wait for space is feasible. If not, control is passed to the Nonrecovery Error routine, and the job is terminated.

WAIT: If the operator uses this operand, the processing is similar to that used in waiting for additional devices to become available (see "Lack of Available Devices"). The Allocation-Wait-for-Unallocation routine issues two DEQ macro instructions to allow access of the UCBs by the Termination routine, by the SYSOUT writer, and by the Extended External Action routine, which is waiting to verify that the correct volumes containing old data sets have been mounted. Next the Allocation-Wait-for-Unallocation routine issues a WAIT macro instruction. When it regains control it determines the source of the POST, and if the job was canceled it loads an error code into register 15 and passes control back to the Wait-for-Space-Decision routine, which passes control to the Nonrecovery routine. If a job step has terminated or a system output writer has deleted a system output-data set, control is passed to the Space Request routine, which repeats its requests for space.

Device Strikeout

The Device Strikeout routine (module IEFX300A) is entered from the Demand Allocation and Decision Allocation routines when a unit has been selected for allocation to a data set. The routine completes the allocation to that data set (and to all related data sets)[1], makes the unit unavailable to other data sets, and returns to its caller.

When it is first entered, the Device Strikeout routine completes the allocation of the selected unit by storing the UCB address in the allocate volume table entry pointed to by the allocate work table (AWT) entry.

If volume affinity is specified, the allocate volume table entries in the group are already chained together. If this is the case, the Device Strikeout routine stores the UCB address in each allocate volume table entry in the chain.

If unit affinity, split cylinder or suballocation is specified, the AWT entries of the data sets in the group are chained together. If this is the case, the Device Strikeout routine stores the UCB address in the first allocate volume table entry for each data set in the chain.

When the Device Strikeout routine stores the UCB address in an allocate volume table entry, it turns off the bits that correspond to that device in the primary and secondary bit pattern fields of the AWT entry. In addition, when it has completed the allocation of a device it determines whether the requested volume is a specific, private, or public volume. If the request was for a specific volume, the non-shareable bit has been set in the AWT. When the routine processes a non-shareable request it sets the appropriate bits off in the bit pattern of all other non-shareable requests. If the volume is private, the routine sets the appropriate bits off in the bit pattern fields of all unallocated data sets' AWT entries. If the volume is public, the bits are turned off for all data sets requesting private volumes.

Separation Strikeout

The Separation Strikeout routine (module IEFXH000) is entered from the Decision Allocation routine after a device has been allocated to a data set, and from the TIOT Construction routine when the most suitable device has been selected. The routine implements other data sets' channel and unit separation requests; it sets bits to zero in the primary bit patterns of their AWT entries if the allocation of the corresponding devices would violate separation or affinity requests.

The effect of executing the Separation Strikeout routine is as follows:

- If a data set requests separation from a data set, the devices that would violate that request are removed from consideration.

- A data set that requests affinity with another data set may or may not be allocated the same device (or channel). It will not however, be allocated any device that violates a separation request of the other data set.

The following is repeated for all data sets that request separation from the original data set:

If any data sets have requested channel affinity to a data set that has in turn, requested channel separation from the original data set, the routine sets to zero any bits in their primary bit patterns that correspond to devices on the channel allocated to the original data set.

If any data sets have requested channel affinity to the original data set, the routine sets to zero all bits in their primary bit patterns except those bits that represent devices on the same channel as the unit allocated to the original data set.

If any data sets have requested unit separation from the original data set, the routine sets to zero the bits in their bit patterns that correspond to the device allocated to the original data set.

If any data sets have requested unit affinity to a data set that has, in turn, requested unit separation from the original data set, the routine sets to zero the bits in their primary bit patterns that correspond to the device allocated to the original data set.

Unsolicited Device End Posting

The Unsolicited Device End Posting routine (module IEFVPOST) is a nucleus-resident routine that is entered from the I/O supervisor if an unsolicited device end occurs on a device for which there is a volume to be mounted and verified. The I/O supervisor uses the allocation attention table index in the device's UCB to pass control to the Unsolicited Device End Posting routine.

After receiving control, the Unsolicited Device End Posting routine searches the allocation/termination communication area (ATCA) for the address of a mount verification communication area (MVCA), which indicates the existence of an ECB

----

[1] Related data sets are those with unit or volume affinity, split cylinder, or suballocation relationships.

chain. Then the routine obtains the address of the MVCA extension (MVCAX) and passes the device-end ECB in the MVCAX to the system post routine for posting. The routine repeats this processing until the last MVCA is processed, and therefore, the end of the ECB chain is reached.

If there are no ECBs to be posted, or when the last ECB is posted, the routine returns control to the I/O supervisor.

ALLOCATION EXIT

The Allocation Exit routine (module IEFSD41Q) is entered from the Extended External Action routine after all volumes have been verified and allocation for a task is complete, or from other allocation modules when an error condition has been detected.

If the routine is entered because of an error condition, it sets the step-flush bit (if the error was a condition code error) or job-flush bit on, to initiate flush mode processing. If a TIOT has not yet been constructed, it then passes control to the JFCB Housekeeping control routine. If a TIOT has been constructed, it frees the main storage obtained for the UCB lookup table, then returns control to the calling routine.

When the routine is entered at the completion of the allocation process, it uses the Queue Management Write routine to place the SMB containing allocation messages in the proper output queue if MSGLEVEL=(X,1). If a system output data set exists in the message class, the routine causes the SMB to point to the first DSB in the entry.

If no message class system output data set exists, the routine places the address of the last allocation SMB in the write-to-programmer control block (WTPCB) for WTP processing. However, if a system output data set exists, the routine places the address of the SMB assigned for termination in the WTPCB and sets bit 2 of the WTPFLGSA field to one. Then the routine places the address of the message class QMPA in the WTPCB.

When the routine is entered during the first step of a job, it assigns two SMBs from the SYS1.SYSJOBQE data set and reserves them for WTP processing. The routine places the address of the first reserved WTP SMB in the WTPCB and enters it in the job queue with a pointer to the second reserved WTP SMB.

Write-to-programmer messages that are issued after a checkpoint will be duplicated during automatic checkpoint/restart. Therefore, if the Allocation Exit rotuine is entered during checkpoint/restart processing, it rechains the WTP SMBs previously existing for the step and reinitializes the WTPCB to reflect this status.

In any case, the routine issues two DEQ macro instructions to allow access to the UCBs by Allocation and Termination routines performing other tasks, determines whether allocation took place in the flush mode, and loads the appropriate return code before returning control to the calling routine.

Task Termination

The Termination routine is a collection of modules used as a subroutine by the initiator and by the System Restart routines. If performs the following major functions:

- Termination housekeeping, which includes interlocking access to UCBs and obtaining an SMB for termination messages to the programmer.

- Step termination, which includes disposing of data sets, unallocating I/O devices, processing condition codes, causing the User's Accounting routine to be executed, determining whether the step is to be restarted, and, if there are additional steps in the job, releasing the UCB interlock and returning control to the caller.

- Job termination, which is performed only when the last step of a job (or a system task) is terminated, includes causing the User's Accounting routine to be executed, performing disposition and unallocation processing that could not be done at step termination, writing remaining system messages specified for DSO processing, completing the job's output queue entry, releasing the UCB interlock, and returning control to the caller.

The termination routine is entered under one of the following conditions:

- It is entered via a Branch and Link instruction from the Step Delete routine of the Initiator when a job step has returned control to the initiator.

- It is entered via a Branch and Link instruction from the Alternate Step Delete routine of the initiator when the initiator has determined that a job step cannot be run.

- It is entered via a LINK macro instruction from the Restart Determination routine of the System Restart routine if a job step was started but not completed when the system failure occurred.

When it has completed its processing, the Termination routine returns control to its caller.

TERMINATION HOUSEKEEPING

The Termination Entry routine (module IEFSD42Q) is given control whenever the Termination routine is entered. It provides an SMB for termination messages to the programmer.

When it is entered, the Termination Entry routine obtains main storage for an SMB. From the step control table (SCT), it obtains the address of the last output queue record assigned for an SMB for the current job, and places that address in the main storage area. If the current step has no data sets designated as system output data sets of the message class, the address represents a partially filled SMB. This SMB, which was placed in the output queue during the execution of the Allocation Exit routine, is read into the main storage area, where it will be filled with termination messages to the programmer before being written into the queue. However, if write-to-programmer processing was invoked during execution of the job step, the partially-filled SMB will point to the first WTP SMB for the step. Therefore, the Termination Entry routine will follow the chain of WTP SMBs to find the last one for the step, read it into the main storage area, and place in it the address of the SMB that will be filled with termination messages.

If, however, the current step does have data sets in the message class, the address obtained from the SCT represents a record in the queue reserved for an SMB, but unused unless write-to-programmer processing was invoked during the job step. The partially filled SMB that was written back into the queue during execution of the Allocation Exit routine points to the first data set block (DSB) corresponding to a data set in the message class; the last DSB (and the SCT) point to the record reserved for a termination SMB or the first WTP SMB. Thus, when the message class queue entry is processed by a system output writer, the interpreter and allocation messages will precede the step's data sets; the data sets will be followed by the write-to-programmer messages, if any, and then the termination messages.

The Termination Entry routine also saves the address of the first WTP SMB in the SCTCRWTP field of the SCT and places a count of the WTP SMBs for the step in the SCTCRCNT field. The information in these fields is used when the job step undergoes an automatic checkpoint restart.

If the problem program ended abnormally, (ABEND), the Termination Entry routine passes control to the Indicative Dump routine (module IEFIDUMP). The Indicative Dump routine writes a message to the programmer indicating the user and system completion codes, and passes control to the Step Termination Control routine.

STEP TERMINATION

When the Step Termination Control routine (module IEFYNIMP) is entered it determines whether the step was normally terminated. If so, it sets the restart fields in the JCT to zero, and stores the queue address of the SCT for the next step in the JCT.

If the step terminated abnormally, however, the routine passes control to the Restart Preparation routine, which determines whether the step is to be restarted, and if so, prepares for the restart. On the return from the Restart Preparation routine, the Step Termination Control routine determines if an automatic checkpoint/restart is in effect. If not, the routine places zeroes in the SCTCRWTP and SCTCRCNT fields of the SCT.

Next, or if the step was normally terminated, the Step Termination Control routine passes control to the Data Set Driver routine for disposition and unallocation processing, and on the return, it tests for error conditions and sets the appropriate bits as follows:

- It tests the Cancel ECB in the Initiator CSCB to determine whether the operator has canceled the job. If so, the ECB has been posted; the Step Termination Control routine sets the job-failed bit in the JCT and uses the WTO macro instruction to inform the operator.

- If the operator has not canceled the job, the routine tests the error code field in the LCT to determine whether the step was abnormally terminated during initiator processing (because, for example, main storage was not available for the initiator, or because of an uncorrectable I/O error). If the step was abnormally terminated, the Step Termination Control routine sets the job-failed bit in the JCT and places a message in the termination SMB.

- If neither of these conditions exists, the routine tests the ABTERM bit in the job step TCB to determine whether the step abnormally terminated during execution. If so, the routine sets theABEND bit in the job status indicators field of the JCT and uses the WTO macro instruction to inform the operator.

When it has tested for these conditions and set the appropriate bits, the Step Termination Control routine frees the main storage occupied by the job step TIOT and determines whether the current step is the last step in the job. If so, it passes control to the User Accounting Linkage routine (which passes control to the user's accounting routine), and on the return it passes control to the Job Termination Control routine. If there are more steps in the job, the Step Termination Control routine passes control to the Job Statement Condition Code Processor routine, which passes control to the User Accounting Linkage routine, then to the Step Termination Exit routine.[1] Even if the job has been canceled on the step abnormally terminated, the remaining steps must be processed (in flush mode). The flush mode processing will insure that unused system input data sets will be deleted, and all system output will be processed.

SMF Processing

If SMF is included in the system, the User Accounting Linkage routine (module IEFACTLK) performs SMF data collection functions as well as furnishing linkage to the user's accounting routine. If the step is the last step of the job, the routine is entered from the Step Termination Control routine (module IEFYNIMP), and again from the Job Termination Control routine (module IEFZAJB3). For all other steps, it is entered from the Job Statement Condition Code Processor routine (module IEFVJIMP).

The processing performed is similar whenever the routine is entered. When it is entered at step termination, the routine reads the step ACT into main storage; when it is entered at job termination, the routine obtains the job ACT. In either case, it determines the maximum amount of main storage used (in hierarchy 0 and 1), then branches to the SMF Termination Record Construction routine.

The SMF Termination Record Construction routine (module IEFSMFWI) builds the SMF job termination record (type 5) or the SMF step termination record (type 4). If the terminating job is a TSO job (terminated via LOGOFF), the routine overlays portions of the type 4 or 5 record with TSO information to convert the record to a TSO step termination record (type 34) or a TSO LOGOFF record (type 35). (The System Management Facilities publication describes the SMF records.) If a non-TSO job is being terminated, the routine also constructs the sign on and sign off messages, and places them in an SMB, before returning control to the User Accounting Linkage routine.

On the return, the Linkage routine constructs a parameter list consisting of 4-byte addresses (on fullword boundaries). The addresses point to the following fields:

1. The job log (the first 36 bytes of the JMR).
2. Step Name (8 bytes).
3. Programmer's name (20 bytes).
4. Job execution time (3 bytes) and number of job accounting fields (1 byte).
5. Job accounting fields.
6. Step execution time (3 bytes) and number of step accounting fields (1 byte).
7. Step accounting fields.
8. Flags (1 byte) and step number (1 byte). If bit 7 of the flag byte is on, the job is to be canceled.
9. Termination status (2 bytes).
10. SMF Step or Job Termination record.

When it has constructed the parameter list, the Linkage routine passes control to the user's accounting routine, which must be named IEFACTRT. On the return, it issues SVC 83 to have the termination record transferred to the SMF buffer, and returns control to its caller.

Restart Preparation

If a job step has been abnormally terminated, the Step Termination Control routine passes control to the Restart Preparation routine (module IEFRPREP). This routine determines whether a restart has been requested and whether it is

[1] If the job-failed bit has been set, the Step Termination Control routine passes control directly to the Step Termination Exit routine.

possible, then requests operator authorization for the restart. If the operator authorizes the restart, the routine sets indicators for data set disposition processing and for re-enqueueing the job's input queue entry.

A restart has been requested, and is possible, only if the following conditions are met:

- MSGLEVEL=1 or (1,1) or (1,0) was specified.
- The RD keyword was specified, and the parameter is either R or RNC, or the RD keyword was not specified, but checkpoints have been taken.
- The ABEND code is one of those specified (when the system was generated) to indicate that a restart is possible or desirable.

If the conditions are met, the Restart Preparation routine uses the WTOR macro instruction to request authorization for the restart. If the operator reply is "YES", the restart is authorized. If the reply is "HOLD" the restart is authorized, but deferred; the routine sets the JCT queue indicator so that the job will be re-enqueued in the hold queue instead of in an input queue. If the reply is "NO", the restart is not authorized.

If the conditions are not met, or if the restart is not authorized, the routine sets the no restart indicators in the JCT and LCT and returns control to the Step Termination Control routine.

If the operator authorizes the restart, the routine determines whether a checkpoint restart or a step restart is to occur. It examines the JCT number of checkpoints and restart indicator fields: if no checkpoints were taken and RD=R or RD=RNC was specified, a step restart is to occur; otherwise, checkpoint restart is to occur.

If a step restart is to occur, the Restart Preparation routine sets bit 5 on in the first byte of the restart switches field in the JCT. (This indicates to the Termination routine that all NEW data sets are to be deleted, and all OLD data sets are to be kept.) The Restart Preparation routine scans the PDQ, decrements all entries for the current step by one, and sets their pass satisfied bits to one. If it encounters an I/O error, the routine issues a message to the operator canceling the restart, and returns control to its caller.

If a checkpoint restart is to be performed, the routine sets bit 4 on in the first byte of the JCT restart switches field. This indicates (to the Termination routine) that all data sets are to be kept.

In either case, the Restart Preparation routine next sets the LCT error field to X'08', and returns control to the Step Termination Control routine.

Disposition and Unallocation

The Step Termination Data Set Driver routine (module IEFYPJB3) is entered from the Step Termination Control routine. If the job step has been abnormally terminated and the allocation message level of the job is zero (MSGLEVEL = (x,0)), the Step Termination Data Set Driver routine reconstructs the allocation messages for the job before it begins to process the SIOTs for the step. It begins SIOT processing by using queue management to read an SIOT for the step into main storage. Control then passes from the Step Termination Data Set Driver routine to the Disposition and Unallocation location routine. The Disposition and Unallocation routine processes the data set specified by the SIOT and returns control to the Step Termination Data Set Driver routine, which brings in the next SIOT. This loop continues until all SIOTs for the step have been processed. Then, the Step Termination Data Set Driver routine returns control to the Step Termination Control routine.

The Disposition and Unallocation routine is divided into two sections: disposition processing (module IEFZGST1), which performs data set dispositions as specified in the DD statement DISP field, and device unallocation processing (module IEFZGST2), which makes the units allocated to the data set available for allocation to other job steps. To interlock UCBs against multiple references during the termination process, these modules issue an ENQ macro instruction when they are entered. Then, to allow use of the Termination routine in the execution of other tasks, the modules issue a DEQ macro instruction.

When a data set has been processed, module IEFZHMSG issues an ENQ macro instruction to protect UCB information and places a message containing the data set name, its disposition, and its volume serials in the termination SMB. If a DISPLAY SPACE command has been issued, bit 5 of byte MSSSB in the master scheduler resident data area is on; if SMF volume information has been requested, bit 4 in the SMCA SMF options field is on. In either case, the routine issues SVC 78 (LSPACE) to have an SMF type 19 record constructed and, if SMF volume information was requested, placed in the SMF data set. If a DISPLAY STATUS command has been issued, the MSDISPST bit in the master scheduler resident data area is on. The routine tests the bit, and if it is on, the routine uses the WTO macro instruction to issue the disposition message to the requesting operator.

Data Set Disposition

When the Step Termination Data Set Driver routine passes control to the Disposition and Unallocation routine, it provides pointers to the TIOT and to the SIOT. If no restart is to occur, the routine performs the disposition specified in the disposition field or (if the job terminated abnormally) the conditional disposition field in the SIOT.

If a restart is to occur, the Restart Preparation routine has set the appropriate bits in the JCT restart switches field. Each time it is to perform a data set disposition, the Disposition and Unallocation routine tests the JCT restart switches field. If a step restart is to occur, it deletes all new data set and keeps all old data sets. If a checkpoint restart is to occur, the routine keeps all data sets.

If no restart is to occur, if there is no disposition specified, and if the data set is not a system output data set, the Disposition and Unallocation routine must assume a disposition based on the status (old or new) of the data set. The routine determines the status by inspecting the status field of the PDQ entry (for a passed data set), or by inspecting the status field of the SIOT. If the routine finds that the data set is "old" (it was created in a previous job), it assumes a KEEP disposition. If it finds that the data set is "new", it assumes a DELETE disposition.

DELETE: If the disposition is DELETE, the Disposition and Unallocation routine first determines whether the data set is cataloged. If the data set is cataloged, and the JFCB Housekeeping routine obtained volume information from the catalog, the Disposition and Unallocation routine issues the UNCATALG macro instruction.[1]

In all cases where the disposition is DELETE, if the devices allocated to the data set are direct access devices, the routine attempts to issue a SCRATCH macro instruction. The SCRATCH can be issued only if all volumes containing the data set can be mounted (either one at a time, or simultaneously). Thus, if any volume containing the data set is not mounted, there must be at least one demountable volume (containing the data set) mounted. If so, the data set is deleted; if not, an error message is issued.

If the data set does not reside on direct access volumes, the volumes containing the data set are demounted, but no SCRATCH macro instruction is issued.

KEEP: If the disposition is KEEP, the Disposition and Unallocation routine issues a message to the operator and passes control to the device availability processing section of the routine.

PASS: If the disposition is PASS, no message is issued to the operator. Control is passed directly to the device unallocation processing section of the routine.

CATLG: If the disposition is CATLG, and the data set is not already cataloged, the Disposition and Unallocation routine issues the CATALOG macro instruction. If, however, the data set is already cataloged, the volume list may have been changed by the Data Management Open, Close, or End-of-Volume routines. If this is the case, the RECATALG macro instruction is issued; if the volume list was not altered, control passes to the device unallocation processing section.

UNCATALG: If the disposition is UNCATALG, the Disposition and Unallocation routine issues the UNCATALG macro instruction, and passes control to the device unallocation section of the routine.

System Output Data Sets: If there is no disposition specified, but the SYSOUT keyword is used in the DD statement, the interpreter uses queue management to assign a record in the appropriate system output queue to a DSB corresponding to the data set. The interpreter places the queue address of the DSB (and of the next DSB in that system output class) in the SIOT of the data set. Control is then passed to the DSB Processing routine (module IEFYTVMS).

The DSB Processing routine determines whether the system output data set contains data. If the DSO indicator bit in the JCT is on, the routine first performs DSO processing. The routine checks the device

----

[1] If the programmer specifies volume information in a DD statement, the JFCB Housekeeping routine will use that information rather than the information in the catalog. The data set will be deleted, but not uncataloged.

type specified for system output. If it is not a direct access device, the routine marks the respective DSB as a dummy SMB to indicate that there is data. Then it issues an ENQ macro instruction (in the shared mode) specifying the DSOCB chain. The routine then turns off the active status bit in each DSOCB used for the step.

Next, the routine checks a bit in the TIOT. If this indicator is on, the data set does contain data. If the indicator is off, a further check is made of a bit in the JCT to determine whether a checkpoint restart might have occurred. If it is possible that a checkpoint restart has taken place, the DSB Processing routine reads the DSCB into main storage. The routine then checks a DSCB field to verify whether the data set contains data.

If the data set does contain data, the routine fills in the DSB for the data set and uses the Queue Management Write routine to place it in the preassigned record. It also sets a bit in the JCT to indicate that a valid data set exists in the system output class in question. If the data set is a dummy (as might be the case if the step has been canceled), a zero DSB is written. A new DSB is created, if, during a system restart procedure, the data set exists and the original DSB has been lost.

If a job step terminates abnormally and a conditional disposition has been specified, the conditional disposition is processed instead of the normal disposition. If a SYSOUT data set contains no data, its disposition is assumed to be DELETE. In any case, the disposition processing is the same.

Unallocating Devices

After the disposition of a data set is determined and processed, the device unallocation portion of the Disposition and Unallocation routine is executed. First, a check is made to determine whether the operator has issued a VARY or UNLOAD command. If so, the status of the device is changed, and a message indicating that the command has been processed is issued to the operator.

When the commands have been executed (if there were any), tests are made to determine whether any of the volumes containing the data set can be demounted. The following kinds of volumes are not demountable:

- Public volumes.
- System residence volumes.
- Volumes mounted on reserved units.
- Permanently resident volumes.
- Volumes whose status is "retain."
- Volumes mounted on system input or system output units.
- Volumes containing data sets having PASS dispositions.
- Volumes whose user count is greater than zero.

Demount messages are issued for any volumes that can be demounted. If the MONITOR DSNAME command is in effect, the demount messages include the names of the first eligible non-temporary data sets associated with the TIOT device entries for which demount messages are to be issued. (See "The MONITOR DSNAME Command.") The addresses of the appropriate UCBs are obtained from the TIOT, and the status of the units used is changed to "allocatable."

If the allocation message level of the job is one (MSGLEVEL = (x,1)), termination messages for the job are written into SMBs. If a job step terminates abnormally, termination messages for the job are issued unconditionally.

When device unallocation processing of a data set is completed, the Disposition and Unallocation routine returns control to the Step Termination Data Set Driver routine. When the last data set has been processed, the Step Termination Data Set Driver routine returns control to the Step Termination Control routine.

JOB Statement Condition Codes

When all SIOTs have been processed, and the SCT has been returned to the input queue, the Step Termination Control routine determines whether there are additional steps to the job. If so, it passes control to the JOB Statement Condition Code routine (module IEFVJIMP). If there were no condition codes specified, the JOB Statement Condition Code routine passes control to the Step Termination Exit routine. If, however, condition codes were specified in the JOB statement, the routine tests the eighth step condition entry to determine whether the step was abnormally terminated. If so, the entry has been set to zeros (except that the ABEND bit in the first byte has been set on) and the routine passes control to the Step Termination Exit routine. If the step was not abnormally terminated, up to eight condition codes in the JCT are compared, in turn, with the step completion code in the SCT (any additional condition codes are ignored).

If any of the condition operators are satisfied by the completion code, the job termination status bit in the LCT and the job-failed bit in the JCT are set on, and the message subroutine (module IEFYSVMS) is used to place a message in the termination SMB. When condition code processing is complete, control is passed to the Step Termination exit routine.

Step Termination Exit

The Step Termination Exit routine (module IEFSD22Q) is the last module of the Step Termination routine to be executed when the current step is not the last step of the job. The routine uses the Queue Management Write routine to place the termination SMB (which points to the first interpreter SMB for the next step) in the output queue, and places the address of the next SCT in the linkage control table (LCT). It issues the POST macro instruction to the ECB in the allocate/IEFVPOST communications block. Finally it loads register 15 with a return code that indicates that there are additional steps in the job, and returns control to the calling routine.

If the current step is the last step in the job, the Step Termination Exit routine is not executed. The Step Termination Control routine passes control to the User's Accounting routine (if it is included in the system), then to the Job Termination Control routine.

JOB TERMINATION

The job termination portion of the Termination routine is executed only when there are no more steps in the job. Its function is to cause the User's Accounting routine to be executed, to complete the job's output queue entries, and to process the following kinds of data sets and volumes, which cannot be processed during step termination:

- Unreceived data sets, which are data sets that were passed, but which were not referred to by subsequent steps.
- Volumes that were retained, but to which reference was never made.

The Job Termination Control routine (module IEFZAJB3) first examines the JCT to determine whether a passed data set queue (PDQ) exists; if so, it uses the Queue Management Read routine to bring each block into main storage. The Job Termination Control routine examines each block, and when it finds an unreceived data set, it passes control to the Disposition and Unallocation routine.

Disposition and Unallocation

When it is entered from the Job Termination Control routine to process unreceived data sets, the Disposition and Unallocation routine finds a pointer to a PDQ block that describes the data set that was passed, but not received.

If the job was abnormally terminated, the job flush bit in the JCT is on, and any conditional disposition specified for the data set will be honored. The Disposition and Unallocation routine uses the Queue Management Read/Write routine to bring the SIOT for the data set into main storage, and the conditional disposition is processed.

If no conditional disposition is specified, or if the job was terminated normally, the disposition performed is either DELETE or KEEP. If the data set was created and passed by the same step (of the current job), the disposition is DELETE; otherwise, a KEEP disposition is performed.

Tape demount messages are issued during data set disposition processing, not during device unallocation. Prior to constructing a KEEP disposition message for a non-temporary data set on a tape volume, the Job Termination Disposition and Unallocation routine extracts the UCB pointer from the PDQ block entry and the volume serial number from the JFCB. If the volume is mounted, termination message module IEFZHMSG extracts the data set name from the JFCB, inserts it into the demount message, and issues a WTO or TPUT (TSO) macro instruction. (See "The MONITOR DSNAME Command.")

In any case, the disposition and unallocation processing is the same performed when entry is from the Step Termination Data Set Driver routine.

When all unreceived data sets have been processed, the volumes that were retained but never referred to must be processed. The Job Termination Control routine again passes control to the Disposition and Unallocation routine, but furnishes no PDQ pointer. This time all UCBs are scanned. The Disposition and Unallocation routine issues demount messages for any demountable volumes on devices whose UCBs contain the task identification found in this initiator's task control block. The routine then returns control to the Job Termination Control routine, which passes control to the User Accounting Linkage routine. This routine performs functions described under "SMF Processing" in the Step Termination portion of this section, then passes control to the Job Termination Exit routine.

Job Termination Exit

The Job Termination Exit routine (module IEFSD31Q) performs functions similar to those performed by the Step Termination Exit routine. It places the last termination SMB in the output queue and posts the ECB in the allocate/IEFVPOST communications block. In addition, it completes the job's output queue entries and issues a job ended or job failed message to the operator as required.

If the DSOCB indicator bit in the JCT is on, the routine issues an ENQ macro instruction (in the shared mode) specifying the DSOCB chain, releases all DSOCBs for the job (to make them available for other jobs), and issues a DEQ macro instruction for the DSOCB chain. If the STOP/MODIFY bit in any DSOCB is on, the routine posts the ECB in the DSOCB to invoke the DSO Stop and Modify Processing routine (module IEFDSOSM). If there is a message class DSOCB for the job, the routine passes control via a LINK macro instruction to the DSO Writer routine (module IEFDSOWR) to process the DSOCB. Upon return, the routine releases the message class DSOCB and deletes the message class SMB from the queue.

The routine uses the Queue Management Read/Write routine to bring the system output class directory (SCD) and the job's first system message block (SMB) into main storage. By matching the output class entries in the SCD (one for each class used by the job) and the SYSOUT class bits in the JCT (set by the SYSOUT Disposition routine), the Job Termination Exit routine determines whether there are output classes other than the MSGCLASS that contain data. If there are system output classes with data, a message showing these classes is constructed and placed in the previously read SMB. Otherwise, the contents of the SMB are changed so the JOB statement will be the first message. The Queue Management Read/Write routine is then used to write the SMB back to the output queue.

After modifying the QMPA according to the information in each SCD entry, the Job Termination Exit routine uses the Queue Management Enqueue routine to enqueue those output classes that contain data and the Queue Management Delete routine to delete those output class entries that contain no data.

There are three bits that must be tested to determine whether a message is to be issued to the operator, and if so, whether a job ended or job failed message is to be issued.[1] If the job notification bit (BAJN) in the master scheduler resident data area is on, and the job-failed and job flush bits in the JCT are off, a job ended message is issued; if the job-failed or job flush bit is on, and the job termination status bit in the LCT is off, a job failed message is issued; otherwise, no message is issued. If the time notification bit is also on, the routine includes the time in the message.

Finally, the routine places a return code in register 15 to indicate that there are no more steps in the job, and returns control to the calling routine.

----

[1] In the case of a TSO job, a different set of bits must be checked. TSO processing is described in the <u>TSO Command Processor PLMs</u> or in the MONITOR commands section of this PLM.

System Management Facilities

The system management facilities (SMF) consist of routines that collect data, exits for user-supplied data collection routines, and routines that record the collected data in a data set. The system management facilities may include either one tape data set (SYS1.MANX) or two direct access data sets (SYS1.MANX and SYS1.MANY). In the latter case, the two data sets are filled alternately: while one is in use, the other may be dumped via the SMF Dump routine (IFASMFDP). This section contains a general description of the system management facilities, and detailed descriptions of the routines that record the data in an SMF data set (the SVC 83, SMF Writer routines, and the SMF Dump routine).

AN OVERVIEW OF SMF

The system management facilities are initialized by routines performing the master scheduling task. The SMF Initialization routine (modules IEESMFIT, IEESMFI3, and IEESMFI2) adds the SMF DD names to the master scheduler TIOT, initializes the system management control area (SMCA) with the SMF parameters, initializes the 10-minute timer, and constructs the SMF IPL and Online Devices records. The SMF Open Initializer routine (module IEESMFOI) initializes the DCBs and JFCBs for the SMF data sets, then uses the ATTACH macro instruction to establish the SMF task.

SMF data collection routines, and exits for user supplied data collection routines, are in the Interpreter, in the Initiator, and in the Termination routine. IBM-supplied data collection routines are in the System Output Writer and the VARY command processor, and other data collection facilities are provided in the supervisor (see the MVT Supervisor PLM).

In the Interpreter, module IEFVH1 of the Interpreter Initialization routine starts construction of the job management record (JMR). The Pre-scan Preparation routine (module IEFVHEB) passes control to the user's JCL Validation routine before each JCL statement is processed, and the Job and Step Enqueue routine (module IEFVHH) passes control to the same user-supplied routine before enqueueing the job. The JMR, now containing data collected by IBM and user-supplied routines, is written in the work queue data set, where it becomes a part of the job's input queue entry.

When the first step of a job is initiated, the Initiator Region Size Determination routine (module IEFSD101) branches to the User Exit Initialization routine (module IEFSMFIE), which constructs the timing control table (TCT) and brings the JMR into main storage. The routine updates the job log portion of the JMR, brings the job accounting control table (ACT) into main storage, and passes control to the user's Job Initiation exit routine.

For all other steps of the job, the User Exit Initialization routine updates the job log, brings in the step ACT, and passes control to the user's Step Initiation exit routine.

Before each step is attached, the Attach routine (module IEFSD263) passes control to the TCTIOT Construction routine (module IEFSMFAT), which updates the TCT and constructs the timing control task input/output table (TCTIOT).

As each step of the job is terminated, the User Accounting Linkage routine (module IEFACTLK) is entered. At step termination it brings in the Step ACT, uses the SMF Termination Record Construction routine to build the SMF step termination record, then passes control to the user's accounting routine. On the return, the routine uses SVC 83 to have the termination record transferred to the SMF buffer.

At job termination, the User Accounting Linkage routine brings the job ACT into main storage. The SMF Termination record construction routine builds the SMF job termination record. The user's accounting routine is executed, and SVC 83 transfers the record to the SMF buffer.

When a VARY ONLINE command is issued, either the VARY/UNLOAD Syntax Scan routine, module IGC1103D (if MCS is not in the system), or the MCS VARY Syntax Check routine, module IGC3303D (if MCS is in the system), passes control to the SMF VARY Record Handler routine (module IEE2303D), which constructs an SMF vary online record and issues SVC 83 to have it transferred to the SMF buffer.

The SMF SVC routine (SVC 83) is used to transfer SMF records to the SMF buffer, to have the contents of the SMF buffer written on the SMF data set, and to initialize and switch data sets. It consists of three load modules: the Record Transfer routine (module IGC0008C), which moves records to the SMF buffer and uses the POST macro instruction to inform the SMF Writer when the buffer is full; the SMF Open routine (module IEESMFOP), which performs data set switching and causes devices to be allocated to the SMF data sets; and the SMF Allocation routine (module IEESMFAL), which performs the allocation.

The SMF Writer (module IEESMFWR) executes the SMF task. The first time it is entered, it issues SVC 83 to have devices allocated to the SMF data sets, and to have the data sets opened. Subsequently, the SMF Writer routine writes records to the SMF data set after making sure that the record will fit. When necessary, it issues SVC 83 to cause data set switching to occur.

THE SMF SVC ROUTINE (SVC 83)

The basic functions of the SMF SVC routine are to transfer the records constructed by data collection routines to the SMF buffer, and when the buffer is full, to have the contents written in the SMF data set. In addition, the SMF SVC routine performs initialization and switching functions: during system initialization, the SVC 83 routine allocates devices to the SMF data sets, and when one of those data sets is full, the routine notifies the operator and switches to the other SMF data set. These functions are performed by three load modules: the Record Transfer routine (module IGC0008C), the Open routine (module IEESMFOP) and the Allocation routine (module IEESMFAL).

The first load of the SVC 83 routine to be entered is always the Record Transfer routine. This routine immediately checks the protection key in the caller's RB to ensure that it is zero, then inspects register 1 (which contains a pointer to the SMCA). If register 1 is positive, the routine is to perform a record transfer. If register 1 is negative, the SVC 83 routine is to perform initializing or data set switching, and the Record Transfer routine uses the XCTL macro instruction to pass control to the SMF Open routine.

Initializing Functions

During system initialization, the SMF Writer routine (module IEESMFWR) issues SVC 83 with register 1 negative. The SVC 83 Record Transfer routine gains control, and because of the negative register 1, passes control to the SVC 83 Open routine.

The Open routine (module IEESMFOP) examines two switches: the Open DS routine switch in the SMCA, which is set on by the Open routine each time it passes control to the SVC 83 Allocation routine; and the first-time switch (in the SMCA miscellaneous indicators field), which is initialized on, and which is turned off by the Open routine when one device has been allocated and the corresponding data set has been opened.

On the first entry to the Open routine, therefore, the first-time switch is on, and the Open DS routine switch is off. The routine loads register 5 with a pointer to the first data set area in the SMCA, and uses the XCTL macro instruction to pass control to the SVC 83 Allocation routine.

The Allocation routine (module IEESMFAL) uses the volume serial or unit address specification as a search argument to find the corresponding UCB. The routine marks the UCB "allocated" and "permanently resident", then sets the Open DS routine switch on and uses the XCTL macro instruction to pass control to the SVC 83 Open routine.

The Open routine (module IEESMFOP) again tests the first-time switch and the Open DS routine switch. This time both are on; the routine turns off the Open DS routine switch and stores a pointer to the UCB in the appropriate master scheduler TIOT entry. Next, the routine turns the first-time switch off, opens the data set, and determines whether the data set is on a tape or direct access volume.

If the data set is on tape, allocation is now complete. The Open routine issues the EXIT macro instruction, and control returns to the supervisor. If however, the data set is on a direct access volume, another device must be allocated for the second SMF data set. The Open routine loads a pointer to the second data set area in the SMCA into register 5, and again passes control to the SVC 83 Allocation routine.

On the return, the Open routine tests the first-time and Open DS routine switches again. This time the first-time switch is off and the Open DS routine switch is on; the routine therefore sets the Open DS routine switch off, opens the data set, and issues the EXIT macro instruction to pass control to the supervisor.

Transferring Records

If the Record Transfer routine (module IGC0008C) is entered with register 1 positive, it transfers a record to the half of the SMF buffer currently in use, and if necessary, has the contents of the buffer written in the SMF data set.

First, the routine tests the miscellaneous indicators field of the SMCA to determine whether writing is allowed, and if so, whether SMF and user records are to be written, or whether only SMF records are to be written.

If no records are to be written, or if only SMF records are to be written and the record is a user record, the SVC 83 Record Transfer routine uses the EXIT macro instruction to pass control to the supervisor.

If the record is to be written, a check is made of the record-type field of the record. For a record that is not either a user record or an SMF record of type 4, 5, or 35, a TIME macro instruction is issued. The resultant time-of-day and date and the system ID from the SMCA are placed in the record. The Record Transfer routine then issues an ENQ macro instruction specifying the SMF buffer and compares the record length to the amount of available space in the SMF buffer. If the record will fit, the routine moves it into the SMF buffer, and increments the SMF buffer physical record length field by the size of the logical record. Then the routine issues a DEQ macro instruction specifying the SMF buffer and uses the EXIT macro instruction to return control to the supervisor.

The routine checks bit 0 of the TCB TSO field to determine whether the current task is a TSO task subject to swapping. If so, the routine obtains the task's TJID from the JSCB and places it in the SMCA. If the task is not subject to swapping, the routine places zeros in the TJID field of the SMCA. This field will be used by the SMF Writer routine when it issues a POST macro instruction to return control after record processing.

If the record is larger than the amount of available space in the SMF buffer, the Record Transfer routine has the contents of the buffer written in the SMF data set: it issues a POST macro instruction specifying the writer ECB, then issues a WAIT macro instruction specifying the buffer ECB (both ECBs are located in the SMCA).

When the buffer ECB has been posted, the SMF Writer has written the contents of the buffer in the SMF data set, and the SVC 83 Record Transfer routine compares the size of the record to the amount of space in the (now empty) SMF buffer. If the record will fit, the Record Transfer routine moves it into the buffer and increments the physical record size field by the size of the record. It issues the DEQ macro instruction, and returns control to the supervisor.

If the record will not fit into the empty buffer, it must be segmented. The Record Transfer routine calculates the number of buffers that will be required to hold the record, and stores that number in the SMCA number-of-record-segments-required field (SMCASGWR). It sets bit 6 on in the SMCA switch A field, then issues a POST macro instruction specifying the writer ECB and a WAIT macro instruction specifying the buffer ECB.

The POST macro instruction causes the SMF Writer to gain control. The Writer routine determines the number of segments that will fit in the remaining space in the SMF data set and if that number is less than or equal to the number of segments to be written, stores it in the SMCA number-of-record-segments-available field (SMCASGFT). The Writer then issues a POST macro instruction specifying the buffer ECB without attempting to write the buffer out.

When it regains control, the Record Transfer routine (module IGC00083) compares the contents of SMCASGWR to the contents of SMCASGFT. If the contents are equal, the record will fit; the Record Transfer routine zeros the two fields, moves the first segment of the record into the buffer, and issues the POST and WAIT macro instructions.

The SMF Writer routine writes the segment in the SMF data set and posts the buffer ECB. When the Record Transfer routine regains control it moves the next segment into the buffer and issues the POST and WAIT macro instructions. This process continues until the Record Transfer routine has moved the last segment of the record into the buffer; it then issues the DEQ macro instruction, and returns control to the supervisor.

If, when the Record Transfer routine compares the two number-of-segments fields, it finds that their contents are not equal, it means that the record will not fit in the data set. The Record Transfer routine tests bit 3 of the appropriate SMCA device status field to determine whether the data set is empty, and if not it requests a data set switch.

The Record Transfer routine requests a data set switch by setting bit 7 in the SMCA switch A field and issuing the POST macro instruction specifying the Writer ECB. It issues the WAIT macro instruction specifying the buffer ECB, and when it regains control the data set switching has been performed.

Although the data set is now an empty data set, the SMCA number-of-record-segments-required field has not been set to zero. The Record Transfer routine sets bit 6 on in the SMCA switch A field and issues the POST and WAIT macro instructions, and the SMF Writer again determines whether the record will fit the data set. When the Record Transfer routine regains control, it performs the comparison again. If the record will fit, the Record Transfer routine causes it to be written, segment by segment.

If the record will not fit into the empty data set, the Record Transfer routine must truncate it. The Record Transfer routine decrements the logical record length field enough so that the record will fit, and causes the record to be written, segment by segment.

In any case, when the last segment of a record has been transferred to the SMF buffer, the Record Transfer routine issues the DEQ macro instruction specifying the buffer and returns control to the supervisor.

Switching Data Sets

When data set switching is required, register 1 is negative. The SVC 83 Record Transfer routine therefore uses the XCTL macro instruction to pass control to the SVC 83 Open routine.

The SVC 83 Open routine (module IEESMFOP) tests the Open DS routine and first-time switches; if the switches are off, data set switching is to be performed, and the routine closes the currently active data set. It uses the WTO macro instruction to inform the operator that the data set is available for dumping, then loads register 5 with a pointer to the other data set area in the SMCA.

Next, the Open routine determines whether the new data set has been opened. If not, it uses the READJ macro instruction to bring the JFCB into main storage and updates the JFCB volume serial and label-type fields with information from the UCB and the SMCA.

The Open routine then uses the OBTAIN macro instruction to determine whether the data set is empty. If it is not empty, the routine sets the "unavailable" bit on in the appropriate device status field in the SMCA and uses the EXIT macro instruction to return control to the supervisor. If the data set is empty, the Open routine issues the OPEN macro instruction to open the data set for BSAM.

When the data set has been opened, the SVC 83 Open routine exchanges the contents of the two data set areas in the SMCA. It uses the WTO macro instruction to inform the operator that the new data set is open, then issues the EXIT macro instruction to return control to the supervisor.

THE SMF WRITER

The SMF Writer routine (module IEESMFWR) is the routine that executes the SMF task. The routine is entered via an ATTACH macro instruction issued in the SMF Open Initializer routine. When it is entered, it issues a WAIT macro instruction specifying the writer ECB in the SMCA. The SMF Writer regains control each time the ECB is posted, and performs one of the following functions:

- It requests opening of SMF data sets.
- It requests data set switching.
- It writes the contents of the SMF buffer in the SMF data set.
- It determines whether the SMF data set contains enough space for a record that must be written in segments.

When it has performed the requested function, the SMF Writer issues a POST macro instruction with the TJID field from the SMCA specifying the buffer ECB in the SMCA. If a foreground job is being handled, the TJID field contains the nonzero TSO terminal job identification number. If a non-foreground job is being handled, the TJID field contains zeros. Finally, the routine issues the WAIT macro instruction specifying the writer ECB. The SMF task is never terminated.

When the SMF Writer gains control after the writer ECB has been posted, it tests bit 7 of the SMCA switch A field. If the bit is on, the SMF Writer has been entered only to request data set switching. It therefore sets register 1 negative and issues SVC 83. On the return, the SMF Writer tests bit 7 again; under these conditions, the bit is on, and the routine issues a POST macro instruction specifying the buffer ECB, and a WAIT macro instruction specifying the writer ECB.

If, when the writer ECB is posted, bit 7 of the SMCA switch A field is off, the SMF Writer routine tests bit 2 of the appropriate device status field in the SMCA to determine whether the SMF data set is on a tape or direct access volume. If bit 2 is off, the data set is on a tape volume, and no space checking is required. The SMF Writer routine initializes that half of the SMF buffer not in use by setting the physical record length field to X'04'. Next, the routine issues the WRITE, CHECK, and TCLOSE macro instructions. Finally, it issues a POST macro instruction specifying the buffer ECB, and a WAIT macro instruction specifying the writer ECB; the SMF task then enters the wait state.

If the SMF data set is on a direct access volume, space checking is required before the record can be written. If the data set contains enough space to hold the contents of the SMF buffer, the SMF Writer issues the WRITE, CHECK, and TCLOSE macro instructions, then issues the POST and WAIT macro instructions.

If the data set does not contain enough space to hold the contents of the SMF buffer, a data set switch is required. The SMF Writer routine issues SVC 83 with register 1 negative; on the return it tests bit 7 of the SMCA switch A field, but since the SMF Writer was not entered only to request data set switching, bit 7 is off. In this case, the space checking is repeated (on the new and empty data set). If the record will fit, the SMF Writer issues the WRITE, CHECK and TCLOSE macro instructions, and finally the POST and WAIT macro instructions.

If bit 6 of the SMCA switch A field is on, the SMF Writer must perform space checking to determine whether the SMF data set contains enough space for an entire segmented record.

The routine first determines whether the data set can contain one buffer load (segment). If so, the routine increments the SMCA number-of-record-segments-allowed field by one, and compares the contents of that field with the contents of the number-of-record-segments-required field. If the contents of the two fields are not equal, the routine determines whether the data set can contain two segments. It continues the process, trying one more segment each time, until it finds either that there is enough space for the required number of segments (the two fields are equal) or that the space is not available. When one of these conditions occurs, the SMF Writer routine issues a POST macro instruction specifying the buffer ECB and a WAIT macro instruction specifying the writer ECB.

THE SMF DUMP ROUTINE

If the SMF data set is on a direct access volume and the data set becomes full, the SMF Writer routine (module IEESMFWR) passes control to the SMF SVC routine, which opens the alternate SMF data set so that recording can continue. The operator then employs the SMF Dump routine (module IFASMFDP) to copy the full data set into the dump data set on tape.

The SMF Dump routine is executed as a job step. It is contained in one assembly module (IFASMFDP), which includes DCB exit routines, a SYNAD exit routine, and an EODAD exit routine as well as the main processing routine.

When the SMF Dump routine is entered, it opens the SYSPRINT (message), DUMPIN (input), and DUMPOUT (output) data sets for BSAM. The DCB exit routines initialize the DUMPIN and DUMPOUT DCB block size and record length fields with default values as necessary. The DUMPIN DCB exit routine also obtains main storage for a pool of buffers and the buffer control table, which contains the addresses of the buffers and the associated DECBs.

When the data sets have been opened, the SMF Dump routine constructs dump header record (a type 2 SMF record as shown in Figure 95), writes it the first record in the output data set, and uses the READ macro instruction to bring an input data set record into each buffer.

The routine issues a series of CHECK macro instructions to check the read operations, and follows each CHECK with a WRITE macro instruction that transfers the contents of the corresponding buffer to the output data set.

When all buffers have been emptied, the routine issues a series of CHECK macro instructions to check the write operations. It follows each CHECK with a READ macro instruction that transfers an input data set record to the buffer whose contents has been written out.

If a permanent I/O error occurs, the SYNAD exit routine issues the SYNADF macro instruction to obtain error information, then writes the information in the SYSPRINT data set and uses the EODAD exit routine to terminate the SMF Dump routine.

When the input data set has been processed, the EODAD exit routine uses the CHECK macro instruction to check the final write operations, then constructs a dump trailer record (a type 3 SMF record as shown in Figure 95) and writes it in the output data set.

When the dump trailer record has been written, or if it is entered from the SYNAD exit routine, the EODAD exit routine frees the main storage obtained for the buffers and buffer control table. If the program has encountered an error, the routine stores a return code of 16 and returns control to the supervisor. If it has encountered no errors, the EODAD exit routine sets the input data set to an unused status by opening the DUMPIN DCB for output and then closing it. Finally, the routine returns control to the supervisor.

THE SMF RECORDS

The IBM-supplied data collection routines collect data elements and assemble them into 31 types of SMF records. Figure 95 shows the data elements that occur in the first 20 record types, and it shows the source from which each data element is obtained.

RECORD TYPE																					ELEMENT NAMES	DATA SOURCE
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	(In order within each record type)	
X	X	X	X	X	X	X	X	X	X	X	X	X		X	X		X	X	X	X	Reserved (value of 0)	
X	X	X	X	X	X	X	X	X	X	X	X	X		X	X		X	X	X	X	Record Type	
X	X	X	X	X	X	X	X	X	X	X	X	X		X	X		X	X	X	X	{ Time Stamp-Time	TIME macro instruction
X	X	X	X	X	X	X	X	X	X	X	X	X		X	X		X	X	X	X	{ Time-Stamp-Date	TIME macro instruction
X	X	X	X	X	X	X	X	X	X	X	X	X		X	X		X	X	X	X	CPU Identification	JMR and SMCA
X																					Job Wait Time Default	SMCA
X																					SMF Buffer Size	SMCA
X																					Main Storage Size	CVT
X																					SMF Options	JCT
	X											X									System Wait Time	SMCA
				X	X	X				X				X	X		X	X		X	Job Name	JMR
				X	X	X				X				X	X		X	X		X	Reader Start Time	JMR
				X	X	X				X				X	X		X	X		X	Reader Start Date	JMR
				X	X	X				X				X	X		X	X		X	User Identification Field	JMR
				X	X																Step Number Field	JMR
						X															Output Class	DSB
				X	X	X															Start Time	JCT for Job and step (dynamic for 6)
				X	X	X															Start Date	JCT for Job and step (dynamic for 6)
				X	X																SYSIN Count	JMR for Job, SCT for step
				X	X																Termination Status	TCB
				X																	Step Dispatching Priority	SCT
					X																Job Selection Priority	JCT
					X																Reader Stop Time	JMR
					X																Reader Stop Date	JMR
					X																Termination Indicator	JCT
					X																SYSOUT Classes	JCT
					X																Checkpoint/Restart Indicator	JCT
					X																Reader Device Class and Type	JMR
					X																Job Input Class	JCT
				X																	Program Name	SCT
				X																	Step Name	SCT
				X																	Requested Region Size	SCT
						X															SYSOUT Count	
						X															I/O Error Indicator	
						X															Data Set Count	
						X															Form Number	DSB
				X																	Main Storage Used	TCT
				X																	LCS Main Storage Used	TCT
				X	X																Storage Protect Key	TCB (PIB in MFT)
				X																	Allocation Start Time	TCT
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		

The format of the ESV and TSO SMF records can be found in the System Management Facilities publication.

Figure 95. SMF Records (Part 1 of 2)

RECORD TYPE																					ELEMENT NAMES (In order within each record type)	DATA SOURCE
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		
				X																	Problem Program Load Time	TCT
				X																	Devices Used	TCTIOT
								X													Devices On Line	UCB
									X	X	X										Devices Varied	Command and UCB
				X	X																Size of Next Fields	
														X	X		X	X		X	Record Indicators	
														X	X						Segment Sizes	
														X	X				X		Reserved (Value of 0)	
														X	X						TIOT Segment	TIOT
														X	X						JFCB Segment	JFCB
														X	X						DCB/DEB Segment	DCB, UCB
														X	X						UCB Segment	UCB, TCTIOT
														X	X						ISAM Extension (ISAM Data Sets Only)	DCB, DEB
														X	X						Tape Extension (Tape Data Sets Only)	
																	X	X			Old Data Set Name	
																		X			New Data Set Name	
																	X	X			Number of Volumes	
																	X	X			Volume Serial List	
																			X		Volume Serial Number	
																			X		Owner Id	Volume Label
																			X		Device Type	UCB
																			X		VTOC Address	
																			X		DS4VTOCI	Format 4 DSCB
																			X		Number of DSCBs	
																			X		Number of Format 0 DSCBs	
																			X		Number of Alternate Tracks	
																			X		Unallocated Space	
																			X		Largest Free Extent	
																			X		Number of Free Extents	
																			X		Reserved	
																				X	Programmer's Name	
				X	X																Problem Program CPU Time	ACT
				X	X															X	Number of Accounting Fields	
				X	X															X	Accounting Fields	
							X														Record Count	SMCA
							X														{ Record Omission Start Time	
							X														{ Record Omission Start Date	SMCA
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		

The format of the ESV and TSO SMF records can be found in the System Management Facilities publication.

Figure 95. SMF Records (Part 2 of 2)

The Write-To-Programmer Facility

The write-to-programmer (WTP) facility consists of three routines that enable the operating system to provide the user with information about his job in the event of an abnormal occurrence during execution. The information is conveyed to the user via messages to the system message class output data set.

The user can invoke WTP by specifying a routing code of 11 in the WTO/WTOR macro instruction (see the MVT Supervisor PLM for a discussion of the Write-to-operator routine). When the Write-to-operator routine (SVC 35 -- module IGC0003E) is entered, it examines all WTO/WTOR messages for a WTP routing code of 11. If such a code is specified WTO passes control to the Write-to-programmer Initialization routine (module IEFWTP00). This routine initializes registers and a work area before passing control to the Write-to-programmer Message Processing routine (module IEFWTP01). The WTP Message Processing routine uses the Transient Queue Management routines (SVC 90) to read, write, and assign SYS1.SYSJOBQE records for WTP messages. If an I/O error occurs in the job queue that is processing the WTP messages or if no record is available for a WTP message, the Write-to-programmer Error Processing routine (module IEFWTP02) receives control.

THE WRITE-TO-PROGRAMMER INITIALIZATION ROUTINE

If a routing code of 11 is specified in the WTO/WTOR macro instruction, the Write-to-operator routine (module IGC0003E) passes control to the Write-to-programmer Initialization routine (module IEFWTP00). Upon entry, the routine determines if a write-to-programmer control block (WTPCB) is initialized for the job. If not, the routine sets register 3 negative to indicate a return from WTP processing and returns control to the WTO routine.

If a WTPCB exists for the job, the routine initializes register 6 with the address of the WTP SMB and sets register 3 negative to indicate a return from WTP processing. Then it checks the job queue for I/O errors. If no errors exist, it determines if the message limit has been reached. If more messages may be processed, the routine obtains and initializes a work area, issues the ENQ macro instruction for the WTPCB, and initializes register with the message length. Then the routine uses the XCTL macro instruction to pass control to the WTP Message Processing routine (module IEFWTP01).

If the WTP initialization routine finds I/O errors when it checks the job queue, it determines the type of exit required by making the decisions shown in Figure 96, then performs the exit.

I/O Errors on Job Queue	Y	Y	Y	Y	Y
WTOR is being processed	Y				
Routing Code Other Than 11 Exists		Y			
MCS included in System			Y	Y	
Routing Code 11 Indicates Message Goes to Console			Y		
Do Action #	1	1	1	2	2

Actions:
1. Set register 3 negative and return control to WTO routine
2. Exit to user

Figure 96. WTP Processing Decision Table

If the limit number of messages has been reached, the routine determines whether a system task or a job step is being performed. If it is a job step, the routine determines the required exit by making the decisions in Figure 96, then exits. If a system task is being performed, the routine obtains and initializes a work area, initializes registers, issues the ENQ macro instruction for the WTPCB, and uses the XCTL macro instruction to pass control to the WTP Message Processing routine (module IEFWTP01).

THE WRITE-TO-PROGRAMMER MESSAGE PROCESSING ROUTINE

The Write-to-programmer Message Processing routine (module IEFWTP01) uses the Transient Queue Management routines (SVC 90) to read, write, and assign records for WTP messages. (For a discussion of the Transient Queue Management Routines, see Part 6 of this manual.) The routine receives control from either the WTP Initialization routine or the WTP Error Processing routine.

When the routine receives control from the WTP Initialization routine, it checks bit 6 of the WTPFLGSA field in the WTPCB. If the bit is one, the routine determines what action to take by making the decision shown in Figure 97.

WTP Invoked For Step	Y	Y	Y	Y	Y	Y
Job Step being processed	Y	Y	Y			
System Task being Processed				Y	Y	Y
Limit Message is Processed	Y					
"No Record" Message is Processed		Y				
Last SMB Used				Y	Y	
New SMB Needed				Y		
Action #	1	1	2	1	2	2

Actions:

1. Free work area issue DEQ macro instruction for the WTPCB, and determine the required exit by making decisions in Figure 96.
2. Use Transient Queue Management Initialization and Read/Write routine to place WTP message in SMB and write it, and if no I/O error occurred using the Transient Queue Management routines, perform action #1 of this table.

Figure 97. WTP Message Processing Routine Decision Table

If bit 6 of the WTPFLGSA field is not one, the routine checks bit 2 of the field to determine if the step already contains system output. If not, it chains the WTP SMB to the last allocation SMB. Then, or if system output already exists, the routine determines if the WTP message limit is reached. If not, the routine performs action #2 in Figure 97.

If the message limit is reached, and a system task is being performed, bit 5 of the WTPFLGSA field is set to one to indicate that the last SMB has been used and action #2 in Figure 97 is performed. However, if the message limit is reached, but a job step is being performed, the routine determines if the last message is processed. If so, the routine frees the work area and determines the required exit by making the decisions in Figure 96. If the last message has not been processed, the routine uses the Transient Queue Management routines to place the message in the SMB and write it, and, if no I/O error occurred using the Transient Queue Management routines, frees the work area and determines the appropriate exit specified by the decisions in Figure 96.

When the WTP Message Processing routine receives control from the WTP Error Processing routine, its processing depends upon whether a system task or a job step is being executed. If a system task is being executed, a WTP system message is added to the SMB containing the "no record" message that was written by the WTP Error Processing routine. The routine uses the Transient Queue Management routines to place the SMB on the job queue. Then it frees the work area, issues a DEQ macro instruction for the WTPCB, and determines the appropriate exit by making the decisions in Figure 96.

However, if a job step is being executed, the routine uses Transient Queue Management routines to write a "no record" WTP message to the WTP SMB, then frees the work area, issues a DEQ macro instruction for the WTPCB, and makes the decisions in Figure 96 to determine the proper exit.

If any I/O errors are encountered while using the Transient Queue Management routines, the routine determines if there are available SMBs. If there are, the routine issues a DEQ macro instruction for the WTPCB and passes control to the WTP Error Processing routine. If there are no available SMBs, however, the routine determines if a "no record" message has been processed; and if not, it passes control to the WTP Error Processing routine. If a "no record" message has already been processed, and there are no reserved SMBs available, the routine frees the work area, issues a DEQ macro instruction for the WTPCB, and makes the decisions in Figure 96 to determine the proper exit. If a "no record" message has been processed, but there are remaining reserved SMBs, the routine places the address of the first remaining reserved SMB in the WTPSMB field of the WTPCB and uses the Transient Queue Management routines to place a WTP message in that SMB. Then, the routine uses Transient Queue Management routines to write the message, frees the work area, issues a DEQ macro instruction for the WTPCB, and makes the decisions in Figure 96 to determine the appropriate exit.

THE WRITE-TO-PROGRAMMER ERROR PROCESSING ROUTINE

The Write-to-programmer Error Processing routine (module IEFWTP02) receives control from the WTP Message Processing routine when I/O errors occur in the job queue that is processing WTP messages or when no record is available for a WTP message.

Upon entry, the routine determines which of the two types of error exists. If there is an I/O error, the routine issues a WTO I/O error message, frees the routine's work area, and sets register 3 negative to indicate a return from WTP processing. Then the routine passes control to the WTO routine (module IGC0003E).

However, if there is no SMB for a WTP message, the routine places the address of the second reserved SMB into the WTPRSMBS field of the WTPCB, issues a "no record" WTO message, and places another "no record" message in the first reserved SMB for WTP. The routine then returns control to the WTP Message Processing routine to finish processing the message.

Appendix A: Tables and Work Areas

INTRODUCTION

This appendix contains descriptions and format diagrams of the major tables and work areas that are used by the MVT Job Management routines. Other tables are described in the body of this publication or in the publication IBM System/360 Operating System: System Control Blocks, GC28-6628. The tables and work areas in this appendix are in alphabetical order, as shown below:

Account Control Table (ACT)
Allocate/IEFVPOST Communication Blocks
Command Scheduling Control Block (CSCB)
Data Set Block (DSB)
Data Set Enqueue (DSENQ) Table
Data Set Name Table (DSNT)
Device Mask Table (DMT)
Device Name Table (DNT)
Direct System Output Control Block (DSOCB)
Initiator Entrance List (IEL)
Initiator Exit List
Initiator Option List
In-stream Procedure Work Area
Interpreter Work Area (IWA)
Job Control Table (JCT)
Job File Control Block (JFCB)
Job File Control Block Extension (JFCBX)
Job Management Record (JMR)
Job Scheduling Entrance List (JSEL)
Job Scheduling Exit List (JSXL)
Job Scheduling Option List (JSOL)
Job Scheduling Work Area (JSWA)
Linkage Control Table (LCT)
Master Scheduler Resident Data Area
Passed Data Set Queue (PDQ)
Procedure Override Table
Step Control Table (SCT)
Step Control Table Extension (SCTX)
Step Input/Output Table (SIOT)
System Management Control Area (SMCA)
System Message Block (SMB)
System Output Class Directory (SCD)
Task Input/Output Table (TIOT)
Timing Control Table (TCT)
Timing Control Task Input/Output Table (TCTIOT)
Volume Table (VOLT)
Write-to-programmer Control Block (WTPCB)

Tables and work areas are shown four or eight bytes wide for convenience, but are not necessarily drawn to scale. The length of each field is given (in bytes) in the upper right corner of the field. Tables that are stored in work queue entries are limited, by convention, to a length of 176 bytes.

The names of most fields are sufficient to describe the fields; those fields that require further explanation are described in the text accompanying the table. Where a macro instruction may be used to include a DSECT of a table in routines using the table, the name of the mapping macro instruction is also given.

Some tables are identified by a one-byte ID field at offset X'03'. These tables and their respective IDs are as follows:

Table ID	Table Name
00	JCT
01	ACT
02	SCT
03	SIOT
05	SMB
06	VOLT
07	DSNT
0A	Procedure Override Table
0C	SCTX
0F	DSENQ
15	DSB
FD(offset X'08')	DSOCB

3 Address in Queue of ACT	1 Table ID=01	4 Address in Queue of Next ACT	
20 Programmer's Name (Job ACT) or Blanks (Step ACT)			
3 Time Required to Run Job or Step	1 Number of Accounting Fields	1 Length of First Accounting Field	Var. First Accounting Field
Var. Other Accounting Fields (if any)	1 Length of Nth Accounting Field	Var. Nth Accounting Field	

Figure 98. Account Control Table (ACT)

ACCOUNT CONTROL TABLE (ACT)

Description: An account control table for each job is created in the interpreter work area by the JOB Statement Processor routine of the interpreter, and placed in the job's input queue entry.

An ACT for each step is created in the interpreter work area by the EXEC Statement Processor routine of the interpreter, and placed in the job's input queue entry.

Job and step ACTs are made available to the User's Accounting routine at step and job termination.

Mapping Macro Instruction: IEFAJCTB

Allocation/Termination Communication Area (ATCA)

Offset					
Hex	Dec				
0	0	Reserved	4	Address of Current MVCA or Zero (0)	4
8	8	ECB for Allocation/Termination	4	Address of First MVCA or Zero (0) in ECB List	4

Mount Verification Communication Area (MVCA)

Offset							
Hex	Dec						
0	0	MVCAX Address	4	Reserved	2	Terminal Job ID (TSO) or Zero (0)	2
8	8	Address of Next MVCA or Zero (0)	4				

Mount Verification Communication Area Extension (MVCAX)

Offset										
Hex	Dec									
0	0	Device-end ECB			4	Device-end ECB Address			4	
8	8	X'80'	1	CANCEL ECB Address	3	Length of This MVCAX	2	Switch Area	2	
10	16	DOM/UCB List Address			4	First DOM ID			4	MCS Only
18	24	Additional DOM IDs							Var.	
		First UCB Address	2	Additional UCB Addresses					Var.	

Figure 99. Allocation/IEFVPOST Communication Blocks

ALLOCATION/IEFVPOST COMMUNICATION BLOCKS

Description: The allocation/termination communication area (ATCA) is resident in the nucleus. The I/O Device Allocation routine issues a WAIT macro instruction specifying the ECB when it must wait for devices to become ready; the Termination routine issues a POST macro instruction specifying the ECB after the devices have been released.

The External Action routine of the I/O Device Allocation routine places the address of the mount verification communication area (MVCA) in the ATCA whenever a mount request is issued for a volume whose verification is delayed until the Extended External Action routine. (Verification is delayed for those volumes that do not require space allocation, i.e., they contain old data sets.) The External Action routine obtains main storage for the MVCA from the system queue area (SQA) and for the MVCA extension (MVCAX) from subpool 0.

The External Action and Extended External Action routines use the MVCA and MVCAX during volume verification processing. Whenever the routines must wait for a device to become ready, they place the MVCA in the IEFVPOST ECB list. After verification is complete, the routines free the main storage occupied by the MVCA and the MVCAX.

The following field in the MVCAX requires explanation:

Switches -- the bits of this field indicate the following when set to one:

Bit	Meaning
0	DOM list exists.
1	MONITOR DSNAME is active.
2	MVCA is in ECB list.
3	Volume requires verification by Extended External Action routine.
4-15	Reserved.

Mapping Macro Instruction: IEFATCOM

Input CSCB

Offset Hex	Offset Dec	Field	Length	
0	0	Address of the Next CSCB in the Chain	4	Header
		Verb Code	1	Header
		Size of CSCB (in doublewords)	1	Header
		Status Flags	1	Header
		Activity Flags	1	Header
8	8	Unused	21	
		UCMI	1	
		Unused	2	
20	32	Command Image	124	
		Reserved	4	
A0	160	Used by MFT only	4	
		Unique Counter for Interpreter	2	
		Express CANCEL SYSOUT	1	
		Used by MFT only	1	
A8	168	Reserved	8	

Offset Hex	Offset Dec	Field	Length	
0	0	Pointer to Next CSCB on Chain	4	
4	4	Verb Code	1	
		CSCB Size in Doublewords	1	
		Status Flags	1	
		Activity Flags	1	
8	8	ID of Started Task (Task's Stepname) or Jobname of Executed Job	8	
10	16	Procedure Name of Started Task (Task's Jobname) or Jobname of Executed Job	8	
18	24	Unit Addr of Device Assigned to Procedure (Started Task only)	3	
		CIB Count Field	1	
1C	28	Protect Key	1	
		UCMI	1	
		TSO Terminal ID	2	
20	32	Pointer to STOP/MODIFY ECB	4	Communication Area
24	36	Pointer to CIB	4	Communication Area
28	40	Reset Priority of Job	1	
		Reserved	7	4
30	48	STOP/MODIFY ECB	4	
34	52	CANCEL ECB	4	
38	56	Reserved	8	
40	64	Communication Switches	1	
		Used by MFT only	11	4
4C	76	JCLS Pointer or In-storage JCT Pointer or JCT TTR	4	
50	80	Queue Manager Parameter List (Input Queue)	36	
74	116	Queue Manager Parameter List (Output Queue)	36	
98	152	Used by MFT only	8	
A0	160	Reserved	16	

Figure 100. Command Scheduling Control Block (CSCB)

COMMAND SCHEDULING CONTROL BLOCK (CSCB)

Description: A command scheduling control block (CSCB) is an area for communications between the Command Scheduling routine (SVC34) and the Command Execution routines. CSCBs are created (in the input format shown above) by several system routines. When a CSCB is created, it is placed in a chain of CSCBs by the command scheduling routine. It remains in the chain until it is deleted from the chain by the Command Scheduling routine, which may also free the main storage occupied by the CSCB. A CSCB is created under the following circumstances:

- A CSCB is created by the Command Scheduling routine each time a task-creating command is encountered. If the task is a reading, writing, initiating, or MOUNT command task, the CSCB is deleted from the chain, and its main storage released, when the task terminates. If the task is a DISPLAY A, or queue manipulation task, the CSCB is deleted from the chain when the task is attached. The main storage occupied by the CSCB is released before the task is terminated.

- A CSCB is created by the Queue Management Dequeue routine each time the initiator dequeues a job. This CSCB is deleted from the chain, and its main storage released, when the last step of the job has terminated.

- A CSCB is created by a system output writer each time it encounters a DSB that was not preceded by another DSB in the current queue entry. The CSCB serves as a communications area, allowing the cancellation (by operator command) of the subtasks established by the writer. The CSCB is deleted from the chain, and its main storage released, when the writer encounters an SMB (or the last block in the current queue entry).

A CSCB is updated (and changed to the control format shown above if necessary) by the Command Scheduling routine when a CANCEL jobname (job selected), CANCEL writer device, MODIFY, or non-periodic STOP command is encountered.

Although most of the fields are self-explanatory, the following require further description:

- Status Flags: This byte indicates the status (pending/not pending) of the CSCB, and the action to be taken by the Command Scheduling routine.

  The bits, when set to one, have the following meaning:

Bit	Meaning
0	Assignment pending.
1	System task CSCB.
2	Cancel all SYSOUT.
3	Insufficient queue space for initiation (ABEND will result).
4	Add this CSCB to chain.
5	Delete this CSCB from chain.
6	Free main storage occupied by this CSCB.
7	Branch to ABTERM.

- Activity Flags: This byte indicates the type of activity with which the CSCB is associated.

  The bits, when set to one, have the following meaning:

Bit	Meaning
0	Job qualifies to be swapped (TSO is in system).
1	TSO terminal job.
2-3	Reserved.
4	Task qualifies to be canceled.
5	Cancel communication switch.
6-7	Used by MFT only.

- Reset Priority: This field contains the priority of a job whose priority has been reset during execution.

- Communication Switches: This byte is used for communication between the routines using the CSCB. The bits, when set to one, have the following meaning:

Bit	Meaning
0	Reserved.
1	Reader return with in-storage JCT.
2	Writer pause data set.
3	Writer pause forms.
4	ID specified in START command.
5	No data set integrity.
6	Hierarchy 1 specified in START command.
7	Default to hierarchy 0.

Mapping Macro Instruction: IEECHAIN

Queue Address of This DSB 3			Table ID = 15 1	
Queue Address of Next DSB 3			Reserved 1	
Reserved 4				
Reserved 1	Logout Bits 1	Status Byte (DSB='FF') 1	Reserved 1	
Entry Length 1	Status A 1	Pool Location 2		TIOT Entry
DDNAME 8				
Queue Address of JFCB 3			Status C 1	
Status B 1	Reserved 1	UCB Address 2		
Additional TIOT Information V				
Reserved 1	Class Type 1	Reserved 2		
Form Number 4				
Program Name 8				
CPU Identification 4				Job Log
Job Name 8				
Job Entry Time 4				
Job Entry Date 4				
User ID 8				

Figure 101. Data Set Block (DSB)

DATA SET BLOCK (DSB)

Description: Space is reserved in the output queue entry for a DSB when the interpreter encounters a DD statement describing a system output data set. The DSB is filled in by the Termination routine when the step that created the data set is terminated. When the job's output queue entry is processed, each DSB provides the system output writer with a pointer to the JFCB, as well as with information about processing the data set.

The DSB contains the following fields:

- Queue Address of This DSB: This 3-byte field contains the TTR address of this DSB.

- Table ID: This 1-byte field identifies the table as a DSB. It contains a X"15".

- Queue Address of Next DSB: This 3-byte field contains the TTR address of the next record in the queue entry. If this DSB is the last record in the entry, the field contains zeros.

- Logout Bits: If the high-order bit in this 1-byte field is on, the DSB represents the system log data set. The System Output Writer does not delete the system log data set when it has been processed. The remaining bits in this field are reserved.

- Status Byte: This 1-byte field is set to X'FF' to distinguish the DSB from an SMB.

- TIOT Entry: A portion of the DSB consists of information from the TIOT entry for the corresponding data set. The "Additional TIOT Information" field in the entry exists only if the data set resides on two or more units. The field, which is variable in length, contains status A information and UCB addresses. The information for the TIOT Entry portion of the DSB is extracted from the job step TIOT by the Job Termination routine.

- Class Type: This 1-byte field specifies the output class in which the data set is to be processed.

- Form Number: This 4-byte field contains the form number parameter specified with the DD statement SYSOUT keyword.

- Job Log: The job log portion of the DSB contains information about the job that was collected by SMF routines. This information, which is similar to that contained in the job management record (JMR), is described in detail in the discussion of the JMR in this section of this publication.

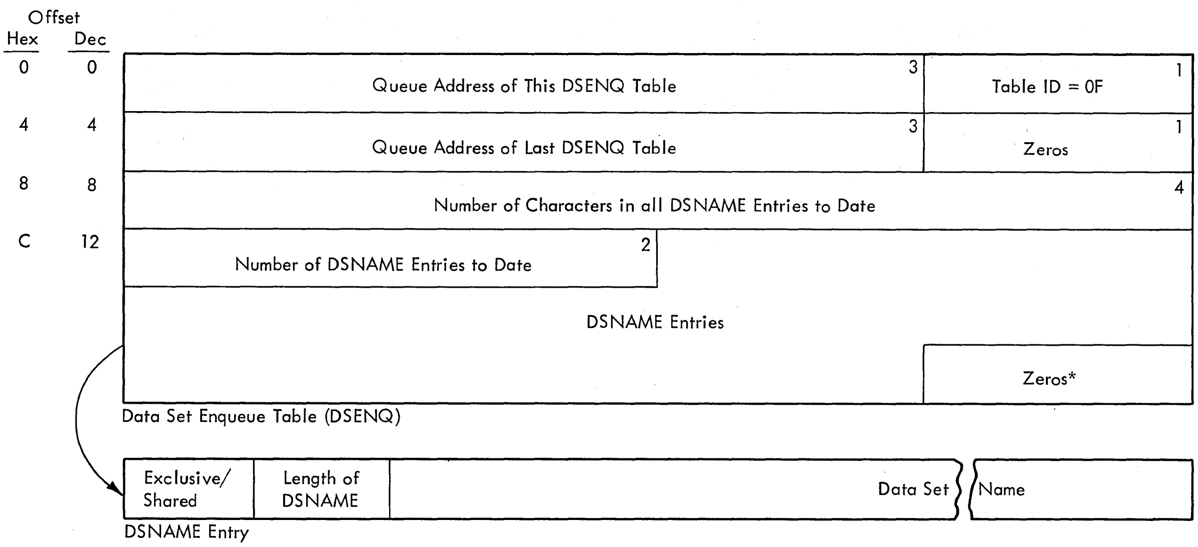

Offset
Hex
Dec
0
0
Queue Address of This DSENQ Table
3
Table ID = 0F
1
4
4
Queue Address of Last DSENQ Table
3
Zeros
1
8
8
Number of Characters in all DSNAME Entries to Date
4
C
12
Number of DSNAME Entries to Date
2
DSNAME Entries
Zeros*
Data Set Enqueue Table (DSENQ)
Exclusive/ Shared
Length of DSNAME
Data Set
Name
DSNAME Entry

Figure 102. Data Set Enqueue Table(DSENQ)

DATA SET ENQUEUE TABLE (DSENQ)

Description: The data set enqueue table (DSENQ) is built by the DD Statement Processor routine of the interpreter, and is used by the initiator to construct an ENQ macro instruction parameter list to prevent routines performing different tasks from using the same exclusive data sets concurrently. The table contains an entry for each data set (except temporary data sets) required for a job.

*If the last entry uses the last available space in the tables but no overflow occurs, the zero bytes are omitted.

3 Queue Address of DSNT	1 Table ID = 07
3 Chain Address	1 Reserved
Var. First DSNAME (1 to 44 Characters)	
Var.	
Var. Nth DSNAME (1 to 44 Characters)	

Data Set Name Table (DSNT)

Figure 103. Data Set Name Table (DSNT)

DATA SET NAME TABLE (DSNT)

Description: The data set name table (DSNT) contains the volume reference data set names for one step as found in the DD statements. The table is created by the DD Statement Processor routine of the interpreter, and placed in the input queue.

The DSNT is pointed to by the step control table (SCT), which also contains a count of the number of bytes in the DSNAMEs for the current step.

Each DSNAME entry is pointed to by the appropriate SIOT until volume resolution (at which time the SIOT points to the VOLT).

The DSNT is used by the JFCB housekeeping routine of the I/O Device Allocation routine to retrieve volume information concerning data sets referred to by data set name in the DD statement VOLUME=REF parameter.

2 Number of Entries	2 Pointer to Mask of Direct-Access Devices

DMT Header

1 Reserved	1 Entry Status	2 Number of Possible Devices
4 Device Type		
Var. Bit Pattern of Possible Devices		

DMT Entry

Figure 104. Device Mask Table (DMT)

DEVICE MASK TABLE (DMT)

Description: The device mask table (DMT) is built and placed in SYS1.LINKLIB when the system is generated. It contains a 4-byte header, and an entry for each unit on the system as well as entries for each group of units described by a generic (IBM-supplied) name, and each group of units described by a user-defined name. In addition, it contains an entry for the group consisting of all direct access devices on the system. During demand allocation, the table is loaded and the entry corresponding to the unit name specified for each data set is found; the device type fields in the appropriate device name table (DNT) entry and DMT are matched, and the bit pattern in the matching DMT entry is placed in the data set's allocate work table (AWT) entry. The DMT is deleted during exit from demand allocation.

Although most of the fields in the device mask table are self-explanatory, the following require further explanation:

- DMT Entry Status: This field indicates whether the units represented in the entry are in the same generic group. If so, the high-order bit in the field is on.
- Device Type: This field is an internal equivalent of the unit name to which the entry corresponds.
- Bit Pattern: This field contains a bit that corresponds to each unit on the system. The field is a minimum of 4 bytes in length; it is increased in size (in 4-byte increments) when the number of units on the system exceeds 32. Each bit in the pattern corresponds, positionally, to a field in the scheduler UCB lookup table (see Figure 89). The bits in the pattern that correspond to units included in the group are set on; all other bits in the pattern are set off.

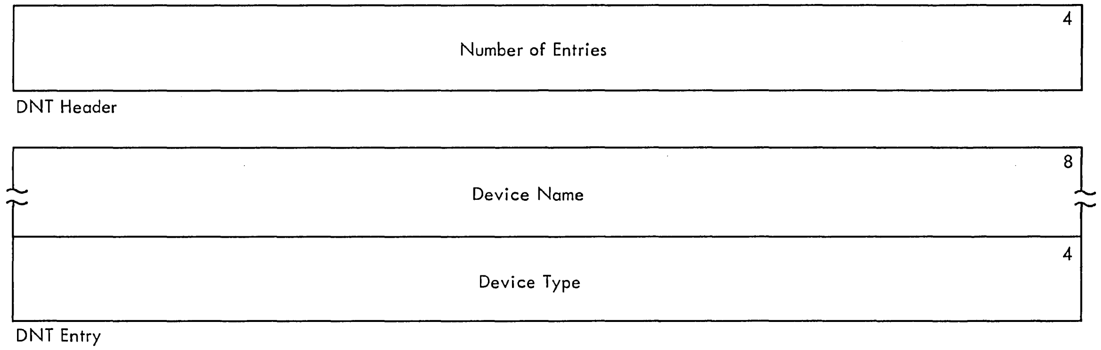


Figure 105. Device Name Table (DNT)

DEVICE NAME TABLE (DNT)

Description: The device name table (DNT), which is built and placed in SYS1.LINKLIB when the system is generated, contains a 4-byte header, and an entry for each unit name in the system. There are three kinds of unit names:

- Specific: A specific unit name is a 3-character, IBM-supplied name, where the first character indicates the channel, the second character indicates the control unit, and the third character indicates the unit. Entries for specific unit names are not placed in the DNT.

- Generic: A generic unit name is an IBM-supplied name of up to eight characters, describing the group that contains all units with the same characteristics on the system. For example, the generic name "2400" describes the group that consists of all of the 2400 series 9-track magnetic tape drives on the system.

- User-Assigned: A user-assigned name is a name supplied by the installation, of one to eight characters in length, that describes a group of one or more units on the system. The composition of the group is defined by the installation

Unit names are left-justified in the device name field, which is padded with blanks.

The DNT is loaded from SYS1.LINKLIB and used during the JFCB Housekeeping portion of the I/O Device Allocation routine to translate the unit name specified on a DD statement into the internal (device type) form. The device type is used during demand allocation to find the appropriate bit pattern in the device mask table (DMT). When unit name conversion is complete, the DNT is deleted.

<table>
<tr><th colspan="2">Offset</th><th colspan="8"></th></tr>
<tr><th>Hex</th><th>Dec</th><th colspan="8"></th></tr>
<tr><td>0</td><td>0</td><td colspan="4">Command ECB 4</td><td colspan="4">Address of Next DSOCB 4</td></tr>
<tr><td>8</td><td>8</td><td>Table ID = X'FD' 1</td><td>SYSOUT Class 1</td><td>User's Protect Key 1</td><td>Reserved 1</td><td>Indicator Byte 1 1</td><td>Indicator Byte 2 1</td><td colspan="2">UCB Address 2</td></tr>
<tr><td>10</td><td>16</td><td colspan="4">Address of TIOT (in SQS) 4</td><td colspan="4">TTR of JFCB 4</td></tr>
<tr><td>18</td><td>24</td><td colspan="8">Job Classes 8</td></tr>
<tr><td>20</td><td>32</td><td colspan="8">Job Separator Module Name 8</td></tr>
<tr><td>28</td><td>40</td><td colspan="8">Reserved 8</td></tr>
</table>

Figure 106. Direct System Output Control Block (DSOCB)

DIRECT SYSTEM OUTPUT CONTROL BLOCK (DSOCB)

Description: The direct system output control block is constructed in the system queue area by the Direct System Output Control routine (IEFDSOCP). The routine initializes the control block with information from the TIOT and the parameter fields in the START command and the EXEC statement. After initialization, the field is chained from a field in the master scheduler resident data area.

Most DSOCB fields are self-explanatory. However, there are certain fields that require further explanation:

- Indicator Byte 1: The bits in this byte have the following meanings when set on:

Bit	Meaning
0	Tape output device. If both bits 0 and 1 are on, output is intended for a printer device but will be placed on tape until a printer device is available. If both bits 0 and 2 are on, output is intended for a punch device but will be placed on tape until a punch device is available.
1	Printer output device.
2	Punch output device.
3	This DSOCB not available for this step.
4	This DSOCB selected for present step.
5	Job separators have been written. (If no job separator module name is entered in the DSOCB, this bit is set on and remains on for all jobs.)
6	The SYSOUT class assigned to the DSOCB is also the current job's message class.
7	This DSOCB is no longer available for selection.

- Indicator Byte 2: The bits in this byte have the following meanings when set on:

Bit	Meaning
0	Internal STOP issued due to scheduler ABEND (MFT only).
1	STOP or MODIFY pending; post ECB. in this DSOCB
2-7	Reserved.

Mapping Macro Instruction: IEFDSOCB

Offset			
Hex	Dec		
0	0	Address of Parameter Field from Initiator's Starting Procedure	4
4	4	Address of ECB/CIB Communication List	4
8	8	Address of Initiator Option List	4
C	12	Address of QMPA	4
10	16	Address of JCT	4
14	20	Address of Initiator Exit List	4

Figure 107. Initiator Entrance List (IEL)

INITIATOR ENTRANCE LIST

Description: The initiator entrance list is a six-word parameter list that is created by the routines calling the Initiator subroutine. It contains pointers to the tables used in communicating information between the caller and the initiator. The fields are described below:

- Address of Parameter Field: This field contains the address of the PARM field from the initiator's starting procedure.
- Address of ECB/CIB Communication List: This field contains the address of the parameter list that contains the STOP/MODIFY ECB pointer and the CIB pointer.
- Address of Initiator Option List: This field contains the address of the parameter list that specifies the caller-defined options for the Initiator subroutine
- Address of QMPA: This field contains the address of the queue management parameter area to be used by the Initiator subroutine.
- Address of JCT: This field contains the address of the job control table of the job that the Initiator soubroutine is to execute.
- Address of Initiator Exit List: This field contains the address of a list of optional exit routines specified by the user.

Mapping Macro Instruction: IEZIEL

Length of Exit List 2	Return Code 1	Reserved 1	Address of CSCB 4	Header
Exit Definition 1	Exit ID 1	Linkage 6		First Data Entry
Exit Definition 1	Exit ID 1	Linkage 6		Nth Data Entry

Figure 108. Initiator Exit List

INITIATOR EXIT LIST

Description: The initiator exit list is built by the Initiator Control routine to provide linkage between the initiator and optional exit routines. The list consists of an 8-byte header entry and a series of 8-byte data entries, each of which describes an exit. The list contains the following fields:

- Return Code: The return code is passed from the initiator to the exit routine and indicates the following:

Code	Meaning
0	Normal completion of task
4	Device allocation error during single-job processing

- Address of CSCB: This field contains the address of the CSCB of the specified task to be initiated.

- Exit Definition: This field specifies the type of linkage to be used between the initiator and the exit routine. Bits 0 and 1 contain the following specifications, bits 2 through 7 are reserved:

Setting	Linkage Type
00	Default; no exit specified
01	Branch to address
10	LINK to name
11	XCTL to name

- Exit Identifier: This field specifies when the exit is to be taken:

Setting	Exit Time
X'80'	Exit upon termination

- Linkage information: The type and format of the linkage information depends on the specification in the Exit Definition field:

Specification	Linkage Information
Name	6-byte EBCDIC name
Address	3-byte address, right justified
V-type address constant	Zeros

Mapping Macro Instruction: IEZIEL

Offset Hex	Offset Dec				
0	0	Length of Option List 2		Option Switches One 1	Options Switches Two 1
4	4	Protect Key 1	Option Switches Three 1	Reserved 2	

Figure 109. Initiator Option List

INITIATOR OPTION LIST

Description: The initiator option list contains the processing options of the initiator. The list is created by the Initiator Control routine. It contains the following fields:

- Length of Option List: This field contains a binary number that is the number of bytes in the option list.
- Option Switches One: This field specifies the following intiator options.

Bit	Meaning
0	If this bit is on (set to one), the initiator uses the protect key specified in the "Protect Key" field. If this bit is off (set to zero), the initiator obtains a nonzero protect key.
1	If this bit is off, the initiator uses dedicated work files.
2	If this bit is on, the initiator does not process STOP or MODIFY commands.
3	If this bit is off, the initiator obtains a minimum size region or greater. If this bit is on, the initiator obtains the specified size region.
4	If this bit is on, the initiator suppresses the CANCEL command except during allocation processing.
5	If this bit is off, the initiator dequeues jobs for processing until a STOP is recognized. If this bit is on, the initiator processes only one job.
6	If this bit is off, the initiator processes the START and MODIFY CIBs and passes an error return if CIB or parameter information is incorrect.
7	Reserved.

- Option Switches Two: This field specifies the following initiator options.

Bit	Meaning
0	If this bit is off, the initiator times job steps as indicated by the "Time" field in the SCT. If this bit is on, the initiator does not time steps.
1	If this bit is off, the initiator permits checkpoint/restart processing if requested. If this bit is on, the initiator does not permit checkpoint restart processing.

2	If this bit is off, the initiator allows direct SYSOUT processing. If this bit is on, the initiator suppresses DSO processing.
3	If this bit is on, the initiator operates in hierarchy zero.
4	If this bit is on, the initiator operates in hierarchy one.
5	If this bit is on, the initiator uses track stacking.
6	If this bit is on, the initiator bypasses allocation recovery.
7	If this bit is on, the initiator does not wait for data sets to become ready.

- Option Switches Three: This field specifies the following initiator options.

Bit	Meaning
0	If this bit is on, the initiator does not wait for job queue space.
1-7	Reserved.

- Protect Key: This field contains the protect key specified by the invoker of the initiator.

Mapping Macro Instruction: IEZIEL

Offset				
Hex	Dec			
0	0	4 Addr of Next Record of Compressed Procedure	4 Addr of In-Stream Procedures QMPA	
8	8	168 Compression Work Area		
B0	176	4 Reserved	1 No. of Entries in Directory	3 Address of Next Record From Assign Routine
B8	184	8 Name of First In-Stream Procedure		
C0	192	3 Addr of First Record of First Procedure	Name of Second In-Stream Procedure	
C8	200	8 Name of Second In-Stream Procedure (Cont)	3 Addr of First Record of Second Procedure	132
		Other Procedure Entries		
			Name of Fifteenth In-Stream Proc	
157	343	8 Name of Fifteenth In-Stream Proc (Cont)	3 Addr of First Record of Fifteenth Procedure	3 Reserved

Figure 110. In-stream Procedure Work Area

IN-STREAM PROCEDURE WORK AREA

Description: The 352-byte in-stream procedure work area is obtained from subpool 0 by the Interpreter Verb Identification routine (module IEFVHCB) when the first in-stream procedure is encountered. The work area consists of two parts: the compression work area used by the Record Compression routine (module IEZNCODE) and the Record Decompression routine (module IEZDCODE) to compress and decompress records and the in-stream procedure directory, which contains the procedure names and addresses of the procedures entered in the input stream. When a job that contains in-stream procedures terminates, the Interpreter Verb routine (module IEFVHCB) releases the storage occupied by the in-stream procedure work area.

Mapping Macro Instruction: WORKAREA

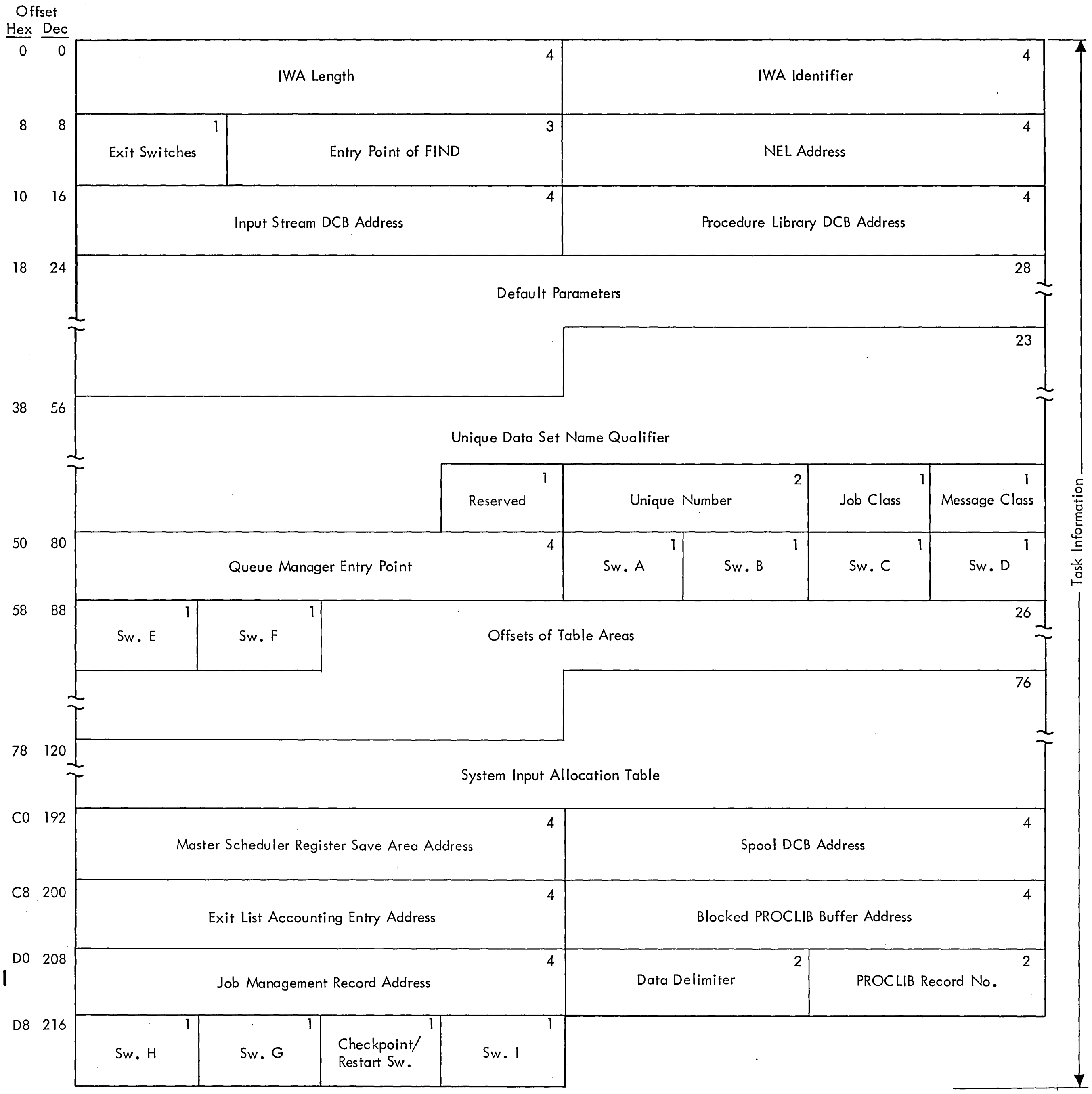


Figure 111. Interpreter Work Area (IWA) -- (Part 1 of 3)

Offset Hex	Dec					
						36
E0	224	Queue Manager Parameter Area (QMPA)				
100	256	Queue Address Table 28				
						Input Stream Statement Parameter List
120	288	Input Stream Statement Parameter List (continued) 8				Procedure Library Statement Parameter List
128	296	Procedure Library Statement Parameter List (continued) 8				48
130	304	Procedure Library Merge Control Data				
						QMPA Address 4
160	352	Address of Parameter List Used for Processing In-Stream Procedures 4				Procedure Referback Dictionary Address 4
168	360	Program Referback Dictionary Address 4				DSENQ Table Address 4
170	368	Referback Dictionary (Input) 176				
220	544	Referback Dictionary (Search) 176				
2D0	720	Job Control Table 176				
380	896	No. of SCTs 1	No. of Job Lib SIOTs 1	Roll Parameter 1	Checkpoint/ Restart Sw. 1	Symbolic Parameter Table Address 4
388	904	Calling Step Name for Restart 8				
390	912	Procedure Step Name for Restart 8				
398	920	SYSCHK DD Statement Address 4				Job Statement Region Size 4
3A0	928	Address of In-Stream Procedure Work Area 4				Queue Address of VOLT 3 | VOLT Length
3A8	936	VOLT Length (continued) 2	Address of BPAM Access Method 3	DD Internal Number 1	Switch X1 1	

Job Information

Figure 111. Interpreter Work Area (IWA) -- (Part 2 of 3)

3B0	944	DD Name Reference Table	78	Step Information
		Reserved	4	
400	1024	Step Control Table	176	
4B0	1200	System Message Block	176	
560	1376	Data Set Name Table	176	
610	1552	Volume Table	176	
6C0	1728	Track Stacking Work Space	4	
		Reserved	28	
6C8	1736			
6E0	1760	Checkpoint/Restart Reinterpretation of JCL	8	
6E8	1768	Sw. Y	1	Statement Information
		Sw. Z	1	
		Return Codes	2	
		Reserved	20	
6F0	1776			
700	1792	Intermediate Text Buffer	176	
7B0	1968	Internal Text Addresses	24	
7C8	1992	Current Register Save Area Address	4	
		Local Work Area Address	4	
7D0	2000	Reserved	4	
		Reserved	4	
7D8	2008	Rollout Save Area	4	
		Reserved	3	
		Sw. Y2	1	
7E0	2016	Reserved	8	
7E8	2024	Reserved	4	Task Information
		NEL JCL Pointer for Post Scan Routine	4	
7F0	2032	Reserved	2	
		*	1	
		**	1	
		***	1	
		****	1	
		MCS Command Authority	2	
7F8	2040	MCS Pointer to Console ID	4	
		Reserved	4	

* Priority Change Value for CHAP Macro

** Default Allocation Message Level

*** Default JCL Message Level

**** Length of Fixed Part of Symbolic Parameter Substitution Message

Figure 111. Interpreter Work Area (IWA) -- (Part 3 of 3)

INTERPRETER WORK AREA (IWA)

Description: The 2048-byte interpreter work area (IWA) is obtained from subpool zero by a GETMAIN macro instruction in module IEFVH1 (the Interpreter Initialization routine). The IWA contains information used by the Interpreter routines; it is the area in which job description tables are built before they are placed in the work queues.

Although most of the fields in the interpreter work area are self-explanatory, the following require further description:

- Exit Switches: A bit is set on in this field to represent each exit specified in the NEL.
- Entry Point of FIND: This field contains the entry point of the special FIND Access Method routine.
- Default Parameters: The PARM Field of the EXEC statement in the reader procedure contains parameters to be used when no explicit specification is made. These parameters specify whether the installation requires a programmer's name or account number on each JOB statement, the priority to be assigned to a job if no priority has been specified, whether commands in the input stream should be processed (or ignored), and the device, primary quantity, and secondary quantity to be allocated to system output data sets.
- Switches A-I: These fields contain internal switches used for communicating status information among the Interpreter routines.
- PROCLIB Record No: The number in this field is the number of logical records of the current procedure library physical record that have been processed.
- System Input Allocation Table: This area contains a list of pointers to the UCBs corresponding to units available for allocation to system input data sets.
- Queue Address Table: This area contains the addresses (in TTR form) of the next two records assigned to the job's input queue entry, and the addresses (in TTR form) of the first joblib SIOT, the first scan dictionary record and the DD override table.
- Input Stream Parameter List: This area describes the statement last encountered in the input stream, and contains a pointer to the field currently being processed.
- Procedure Library Parameter List: This area describes the statement last read from the procedure library, and contains a pointer to the field currently being processed.
- Procedure Library Merge Control Data: This area contains information used in merging statements from the input stream with statements from the procedure library. The information includes the statement names, the step names, and the names of the previous and next procedure steps.
- QMPA Address: This area contains the address of the QMPA for the job, unless an in-stream procedure is being processed. In that case, the address of the in-stream procedure QMPA is placed in this area.
- Address of In-stream Procedure Parameter List: The first byte of this field is a switch as follows:

Bit	Meaning, if on
0	In-stream procedure being processed.
1	End-of-file for reader encountered while module IEFVINA is reading in-stream procedure.
2-7	Reserved.

- Address of BPAM Access Method: This area contains the address of the BPAM access method and is initialized when the first in-stream procedure is being processed.
- IEFVGM Temporary Save Area: This field is used by module IEFVGM to temporarily save a pointer to the QMPA.
- Data Delimiter: The valid two-byte DLM parameter value is saved here.
- Mapping Macro Instruction: IEFVMIWA

Offset Hex	Offset Dec	Field	Length
0	0	Address in Queue of JCT	3
		Table ID = 00	1
		Internal Job Serial Number	1
		Job Status Indicators	1
		Message Class	1
		MSG-LEVEL	
		Job Priority	
8	8	Job Name	8
10	16	Teleprocessing Terminal Name	8
18	24	Address in Queue of PDQ	4
		Address in Queue of GDG Bias Count Table	4
20	32	Address in Queue of First SCT	4
		Address in Queue of First SMB	4
28	40	Address in Queue of Job ACT	4
		Address in Queue of First SCD	4
30	48	Address in Queue of Last DSB	4
		Key (of Track ID) for SMB	2
		First Job Condition Code	2
38	56	First Job Condition Operator	1
		Reserved	1
		Seven Additional Job Condition Codes and Operators	28
50	80	Restart Switches	2
58	88	TTR of DSENQ Table	4
		Region Parameter	2
		Queue Ident	1
		Number of Steps	1
60	96	TTR of Compressed TIOT	4
		Checkpoint Data Set Device Type	4
68	104	TTR of JFCB for Checkpoint Data Set	3
		Number of tracks on SYS 1 JOBQE used by job	1
		Number of Checkpoints	2
		Vol Seq of Data Set	1
		Reserved	1
70	112	TTR of SCT for First Step to Run	4
		Additional Job Status Indicators	1
		Length of Chkpt ID	1
		Checkpoint ID (left Justified, from 1-16 bytes)	16
78	120		
80	128	Queue Address of JMR	
88	136	JMR Address (Cont'd.)	3
		Date Difference	1
		SMF Options	1
		Cancel Flags	1
		Job Time Limit	3
		Step Start Time	
90	144	Step Start Time (Cont'd.)	3
		Job Start Time	3
		Job Start Date	3
98	152	SYSOUT Classes	5
		Address in Queue of First SMB for Direct Sysout Processing	3
A0	160	Reserved	16

Figure 112. Job Control Table (JCT)

JOB CONTROL TABLE (JCT)

Description: The job control table (JCT) is created in the interpreter work area by the Job Statement Processor routine of the interpreter. It contains information from the JOB statement, job status information, and pointers to other tables in the job's input queue entry. When the interpreter has processed all steps of a job, the JCT is written into the input queue; it is read back into main storage by the Initiator Job Selection and Job Delete routines.

Although most of the fields in the job control table are self-explanatory, the following require further description:

- Job Status Indicators: The sixth byte of the JCT indicates the status of the job as shown below:

  Bit 0 is set to 1 if a JOBLIB DD statement is included with the job.
  Bit 1 is set to 1 if the job is flushed.
  Bit 2 is set to 1 if the job step is cancelled by condition codes.
  Bit 3 is set to 1 if the job step is flushed.
  Bit 4 is set to 1 if one or more steps have been abnormally terminated.
  Bit 5 is the job-failed bit. It is set to 1 if an error condition is encountered that causes the job to be terminated.
  Bit 6 is set to 1 if the job includes a cataloged procedure.
  Bit 7 is set to 0 if the job is a setup job; it is set to 1 if the job is a nonsetup job.

- Restart Switches: This two-byte field indicates the type of restart as shown below:

  Byte 1
  Bit 0 is set to 1 to indicate a system restart.
  Bit 3 is set to 1 if a checkpoint has been taken for a step.
  Bit 4 is set to 1 to indicate a checkpoint restart.
  Bit 5 is set to 1 to indicate a step restart.

  Byte 2
  Bit 0 is set to 1 to indicate that a SYSCHK DD statement is present.
  Bit 1 is set to 1 to indicate that the parameter associated with the RD keyword is not NC.
  Bit 2 is set to 1 to indicate that RD=NR.
  Bit 3 is set to 1 to indicate that RD=NC or RD=RNC.
  Bit 4 is set to 1 to indicate that RD=R or RD=RNC.
  Bit 5 is set to 1 to indicate that direct SYSOUT processing is active at the checkpoint.
  Bit 6 is set to 1 to indicate that the job is eligible for direct SYSOUT facilities.
  Bit 7 is set to 1 to indicate that the DSDR Processing step has not successfully ended.

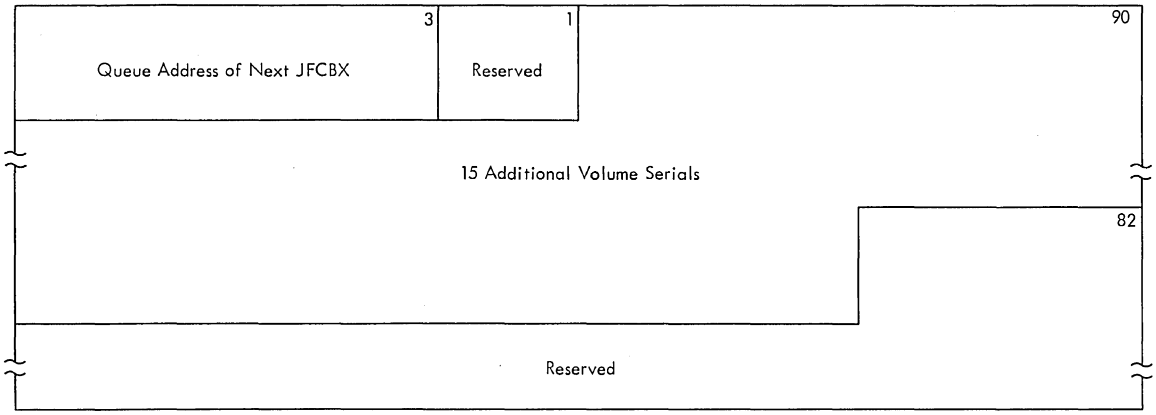
- Additional Job Status Indicators: This byte indicates the status of the job as shown below:
  Bit 0 is set to 1 to indicate SYSIN data for the job.
  Bit 1 is set (to 1) by the Termination routine.
  Bit 2 is set (to 1) to indicate the suppression of warm start messages for this job.
  Bits 3-7 are reserved.

- SYSOUT Classes: The first 36 bits of the five-byte field are used to indicate the system output classes that contain data. The remaining four bits are reserved.

Mapping Macro Instruction: IEFAJCTB

Offset Hex	Offset Dec								
0	0	Data Set Name 44							
						Element Name or Relative Generation Number 4			
30	48	Element Name or Relative Generation Number (continued) 8				J/M + D/M Interface 1	Reserved 3		
38	56	FCB Image-id 4				System Code 3			Reserved 1
		Reserved 2		Label Type 1	Buffer Offset 1	File Sequence Number 2		Volume Sequence Number 2	
48	72	Data Management Mask 8							
50	80	Data Set Creation Date 3			Data Set Expiration Date 3			Indicator Byte 1 1	Indicator Byte 2 1
58	88	Number of Buffers 1	Buffering Technique 1	Buffer Length 2		Error Options 1	Device Characteristics 1	Tape Density 1	Reserved 1
60	96	Reserved 2		Data Set Organization 2		Record Format 1	Option Codes 1	Maximum Block Size 2	
68	104	Logical Record Length 2		Number of Channel Programs 1	Number of Master Index Tracks 1	Relative Location of Key in Logical RCD 2		Reserved 2	
70	112	Reserved 4				Number of Overflow Tracks 1	Number of Volume Serials 1	First Five Volume Serials 30	
						Length of JFCBX 1	Queue Address of First JFCBX 3		
98	152	Primary Quantity 3			Space Type Requested 1	Secondary Quantity 3			Indicator Byte 3 1
A0	160	Directory Quantity 3			Main Storage Address of Split Cylinder JFCB 3			Relative Address of First Track to be Allocated 2	
A8	168	Main Storage Address of SUBALLOC JFCB 3			Average Data Record Length 3			Volume Count 1	Number of Tracks per Cylinder 1

Figure 113. Job File Control Block (JFCB)



Job File Control Block Extension (JFCBX)

Figure 114. Job File Control Block Extension (JFCBX)

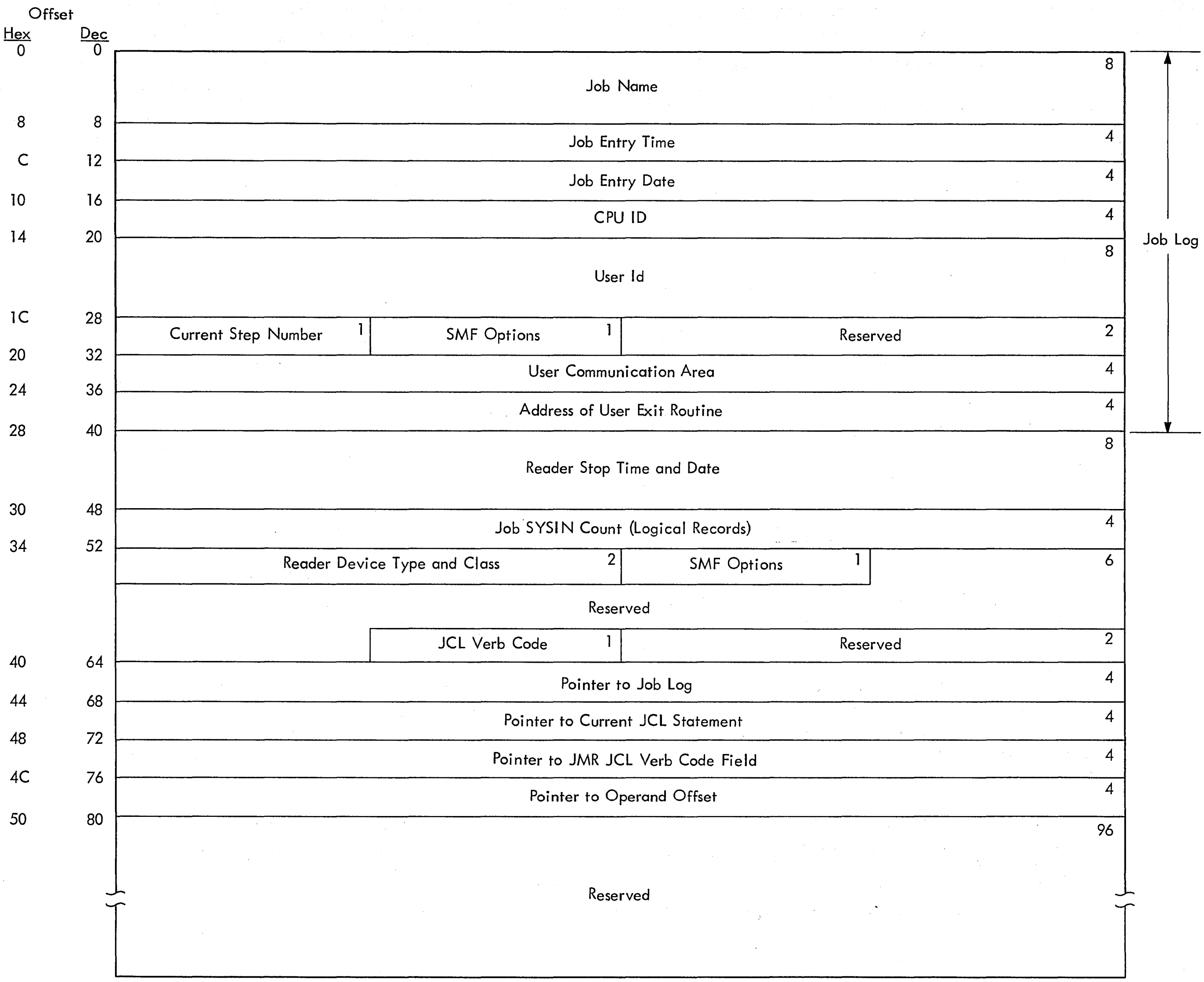
JOB FILE CONTROL BLOCK (JFCB) AND EXTENSION (JFCBX)

Description: A job file control block (JFCB) is constructed in subpool zero (from information in a DD statement) by the Interpreter DD Statement Processor routine. The JFCB is written into the job's input queue entry, and retrieved when a DCB with the corresponding name is opened. The information in the JFCB, which describes the characteristics of a data set, may be modified by the Open routine.

A JFCB contains enough space to record five volume serials. If more than five volume serials are specified, enough job file control block extensions (JFCBXs) to contain the additional volume serials are constructed; each JFCBX can contain up to fifteen additional volume serials.

Additional information on the contents of the JFCB and JFCBX may be found in the publication, IBM System/360 Operating System: System Control Blocks, GC28-6628.

Mapping Macro Instruction: IEFJFCBN



Offset Hex	Offset Dec	Field	Length
0	0	Job Name	8
8	8	Job Entry Time	4
C	12	Job Entry Date	4
10	16	CPU ID	4
14	20	User Id	8
1C	28	Current Step Number	1
		SMF Options	1
		Reserved	2
20	32	User Communication Area	4
24	36	Address of User Exit Routine	4
28	40	Reader Stop Time and Date	8
30	48	Job SYSIN Count (Logical Records)	4
34	52	Reader Device Type and Class	2
		SMF Options	1
		Reserved	6
		JCL Verb Code	1
		Reserved	2
40	64	Pointer to Job Log	4
44	68	Pointer to Current JCL Statement	4
48	72	Pointer to JMR JCL Verb Code Field	4
4C	76	Pointer to Operand Offset	4
50	80	Reserved	96

Figure 115. Job Management Record (JMR)

JOB MANAGEMENT RECORD (JMR)

Description: The Job Management record (JMR) is created as the JCL for a job is interpreted, and continues to be updated during the life of the job. The JMR is written into the job's input queue entry immediately before the JCT is written, and is brought back into main storage when the job is initiated. The JMR contains job information accumulated by IBM-supplied data collection routines; it serves as an information source for SMF records and also contains parameters passed to user exit routines.

The names of some of the fields in the JMR are sufficient to describe their contents. Some fields, however, require further explanation, and these are described below:

- Job Entry Time: This field contains a binary representation of the time (in hundredths of seconds) when the job statement was made available to the Interpreter for scanning.

- Job Entry Date: This field contains a packed decimal representation (00YYDDDF) of the date on which the job was interpreted.

- User Communication Areas: This 4-byte field is available for use by all user exit routines. The contents of this area are not recorded on the SMF data set, nor disturbed by any IBM-supplied routines.

- Reader Stop Time and Date: This field contains a binary representation of the time and a packed decimal representation of the date on which the job was enqueued. If an in-storage access method with a dummy IEFRDER DD statement is specified for the reading task, the reader device type and class field will contain zeros.

- Reader Device Type and Class: This field contains the device type and class of the unit allocated to the reading task, as represented in the UCB device type and device class fields.

- SMF Options: The SMF options field appears at offsets X'1D' and X'36'. This field indicates the options that were selected when the system was loaded or if a TSO job is being processed, the options in this field will apply to TSO processing. The bits, when set to one, indicate that the corresponding options were selected:

Bit	Option
0	Job Accounting
1	Step Accounting
2	User Exits
3	Data Set Accounting
4	Volume Accounting
5	ESV Accounting
6	Temporary Data Set Scratch Records
7	(TSO) Foreground Job

- JCL Verb Code: The value stored in this field indicates the type of JCL statement currently being processed:

Value	Statement Type
0	Null Statement
1	JOB Statement
2	EXEC Statement
4	DD Statement
8	PROC Statement
16	Job is ready to be enqueued

Mapping Macro Instruction: IEFJMR

Offset Hex	Offset Dec		
0	0	Address of EBCDIC Options String (JSELOPT)	4
4	4	Address of JSOL (JSELJSOL)	4
8	8	Address of CSCB for Jobs to be Scheduled (JSELCSCB)	4
C	12	Address of JCLS (JSELJCL)	4
10	16	Address of JSXL (JSELJSXL)	4

Figure 116. Job Scheduling Entrance List (JSEL)

JOB SCHEDULING ENTRANCE LIST

Description: The JSEL is a five-word parameter list that is built by the JCL Edit routine (module IEEVJCL). The JSEL is passed to the Interpreter Control routine (module IEEVRCTL) when the routine is entered to schedule a job step task in a START command or a TSO task from a terminal. The JSEL contains the addresses of tables used to communicate information about the tasks between the caller and the routines that schedule the tasks.

Mapping Macro Instruction: IEFVJSEL

Offset Hex	Offset Dec					
0	0	Length of Exit List (JSXLLGTH) 2	Return Code (JSXLRCOD) 1	Reserved 1	Address of Local Work Area (JSXLCOMM) 4	Header
8	8	Exit Definition (JSXLLINK) 1	Exit ID (JSXLID) 1	Linkage 6		First Data Entry
		Exit Definition 1	Exit ID 1	Linkage 6		Nth Data Entry

Figure 117. Job Scheduling Exit List (JSXL)

JOB SCHEDULING EXIT LIST

Description: The job scheduling exit list is built by the JCL Edit routine (module IEFVJCL) to provide linkage between optional exit routines and the routines that schedule job step tasks from a START command and TSO tasks from a terminal. The list consists of an 8-byte header entry and a series of 8-byte data entries, each of which describes an exit. The list contains the following fields:

- Return Code: The return code indicates the following:

Code	Meaning
0	Return from routines that schedule jobs
1	Return from module IEFVSCAN
2	Return from reader/interpreter
3	Return from initiator

- Exit Definition: This field specifies the type of linkage to be used. Bits 0 and 1 contain the following specifications; bits 2-7 are reserved:

Setting	Meaning
10	LINK to name
11	XCTL to name
01	Branch to address
00	Default; no exit specified

- Exit ID: This field specifies when the exit is to be taken:

Bit	Meaning
0	Post-processing exit routine
1	Initiator pre-invocation exit routine
2	Initiator post-invocation exit routine
3	Reader/interpreter special input access method
4-7	Reserved

- Linkage Information: The type and format of the linkage information depends on the specification in the exit definition field:

Specification	Linkage Information
Name	6-byte EBCDIC name
Address	3-byte address, right justified
V-type address constant	Zeros

Mapping Macro Instruction: IEFVJSXL

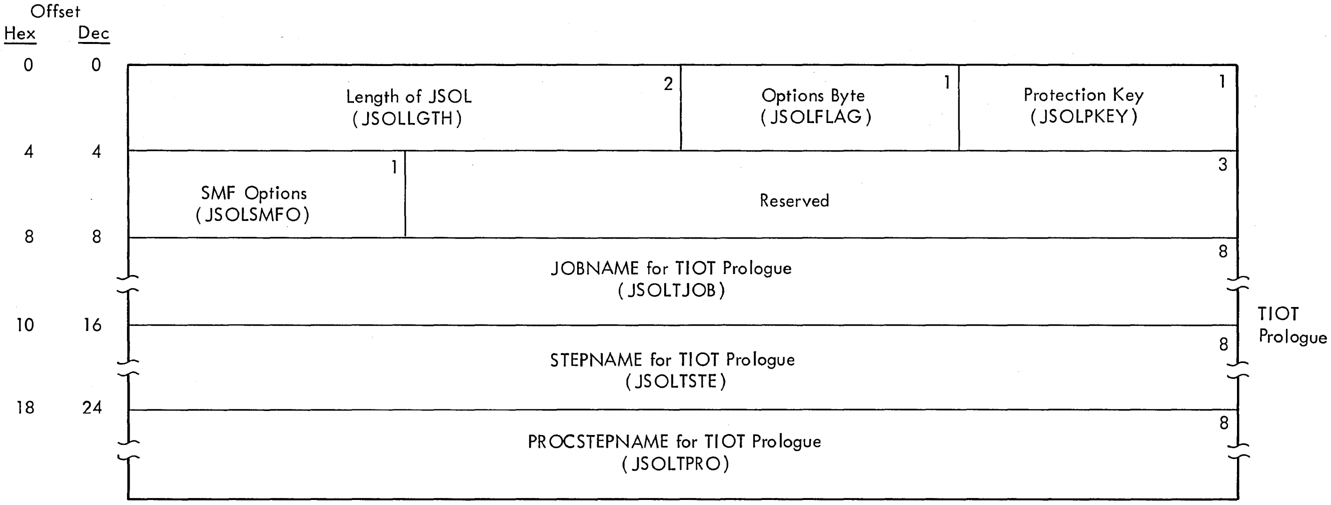

Offset
Hex Dec
0 0
4 4
8 8
10 16
18 24
Length of JSOL (JSOLLGTH) 2
Options Byte (JSOLFLAG) 1
Protection Key (JSOLPKEY) 1
SMF Options (JSOLSMFO) 1
Reserved 3
JOBNAME for TIOT Prologue (JSOLTJOB) 8
STEPNAME for TIOT Prologue (JSOLTSTE) 8
PROCSTEPNAME for TIOT Prologue (JSOLTPRO) 8
TIOT Prologue

Figure 118. Job Scheduling Option List (JSOL)

JOB SCHEDULING OPTION LIST

Description: The job scheduling option list contains the processing options for the routines that schedule job step tasks in a START command and TSO tasks from a terminal. The list is created by the JCL Edit routine (module IEEVJCL). Some of the fields of the JSOL are self-explanatory. The following fields, however, require further description:

- Options Byte: The bits of this field, if set to one, have the following meaning:

Bit	Meaning
0	Use protection key specified
1	SMF is active
2	SMF option byte is in the JSOL
3-7	Reserved

- Protection Key: This field contains the protection key to be used for programs not listed in the linkage table if the protection key option is selected for the task.

- SMF Foreground Options: The bits of this field, if set to one, have the following meanings:

Bit	Meaning
0	Job accounting
1	Step accounting
2	Exits supported
3	Data set accounting
4	Volume accounting
5	ESV accounting
6	Temporary data set scratch record
7	Foreground job

- TIOT Prologue Fields: These fields contain the names to be included in the TIOT prologue that is substituted for the model TIOT.

Mapping Macro Instruction: IEFVJSOL

Offset					
Hex	Dec				
0	0	2 Length of JSWA (JSWALGTH)	1 Options Byte (JSWAOPT)	1 SMF Options (JSWASMF)	4 Address of JSEL (JSWAJSEL)
8	8	(JSWAMOD)			
10	16	4 Address of Track Stack (JSWATSTK)	1 Protection Key (JSWAPKEY)	4 Address of TIOT (JSWATIOT)	
18	24	(JSWAXCTL)			

Figure 119. Job Scheduling Work Area (JSWA)

JOB SCHEDULING WORK AREA

Description: The job scheduling work area contains information about the job step tasks entered into the system via a START command and TSO tasks entered from a TSO terminal. The job scheduling work area is built by the Interpreter Control routine (module IEEVRCTL) in subpool 253; it is used by the interpreter and initiator routines to schedule jobs.

The names of some of the fields are sufficient to describe their contents. However, the fields below require further description.

- Options Byte: The bits of this field, if set to one, have the following meanings:

Bit	Meaning	Flagname
0	Process TERM=TS (TSO)	JSWATERM
1	Use track stacking	JSWATRST
2	Support dedicated work files	JSWAFI
3	Reserved	
4	Use protection key provided	JSWAPKO
5	Bypass allocation recovery	JSWANORC
6	Do not wait for queue space	JSWANOQW
7	Initiator uses RET=USE option for ENQ of data sets; if ENQ fails, job fails	JSWAENQU

- SMF Options: The bits of this field, if set to one, have the following meanings:

Bit	Meaning
0	SMF is active
1	SMF exits are active
2	Timing information is in the protected step control block (TSO)
3-7	Reserved

- Protection Key: This field contains the protection key to be used if the tasks invoked for this job are not in the linkage table and if the protection key option has been specified by the invoker of module IEEVRCTL.

Mapping Macro Instruction: IEFVJSWA

Offset Hex	Offset Dec									
0	0	Limit Pri-Param	1	Address of Job Step CSCB	3	Address of I/O Superv. UCB Lookup Table	4			
8	8	Job Step TCB Address	4	Device Features	1	Linkor's Reg Save Area Addr	3			
10	16	JCT Address	4	SCT Address	4					
18	24	Queue Address of Current SCT	4	Allocate/IEFVPOST Comm. Block Address	4					
20	32	Error Code	4	Communications Area	16					
				Alloc/Terminate Register Save Area Address	4					
38	56	Reserved	1	JFCB House-keeping Indic's	1	Current Step Number	1	Action Code	1	
		Address of Current SMB	4							
40	64	Counter for Assigning Unique Vol. Serials to Passed Data Set Volumes	4	Address of Message Class QMPA	4					
48	72	Return Address to System Task Ctl Rtne	4	Initiator Internal Sw's	1	PARM Field Address	3			
50	80	Timer Work Area	16							
60	96	JOBLIB DCB Address	4	Alloc/Terminate Param List Address	4					
68	104	Register Save Area	144							
F8	248	Queue Management Parameter Area (QMPA)	36							
11C	284	Message Class QMPA	36							
140	320	Number of Buffers	1	Stack Address	3	Queue Breaking Information	4			Track Stacking Information
148	328	ECBLIST Address	4	Initiator Identifier						
150	336	Continued	8	Initiator Force						
158	344	Continued	8	Initiator Limit	1	Force Priority	1	Initiator Priority	1	
		Reserved	1							
160	352	Initiator Options Byte 1	1	Initiator Exit List Addr	3	Initiator Options Byte 2	1	Address of Communications Parameter Area (CPA)	3	
168	360	Initiator Options Byte 3	1	Address of JSCB for Task	3	Reserved	4			
170	368	Reserved	12							

Figure 120. Linkage Control Table (LCT)

LINKAGE CONTROL TABLE (LCT)

Description: The linkage control table (LCT) is built in a main storage area obtained from subpool 253 by the Initiator Initialization routine. It is a communications area used by the routines of the initiator.

Most of the fields in the LCT are self-explanatory; it should be noted, however, that the job termination status bit is the low-order bit of the one-byte device features field.

The following fields require further explanation.

Initiator Internal Switches: When set on, the bits mean:

Bit	Meaning	Flagname
0	Initiator operates in hierarchy one.	LCTIHIER
1	Reserved.	
2	Attach IEFSD0XX.	LCTSD0XX
3	Job flush; use minimum initiator region.	LCTMINRG
4	Taskname not in START command.	LCTSTART
5	Initiator internal stop.	LCTSTOP
6	Job step abnormally terminated (ABEND).	LCTABEND
7	Reserved.	

Initiator Options Byte 1: When set on, the bits mean:

Bit	Meaning	Flagname
0	Initiator uses the protect key specified in the protect key field.	LCTPKEYF
1	Initiator does not use dedicated work files.	LCTDWFF
2	Initiator does not process STOP or MODIFY commands.	LCTSTMDF
3	Initiator gets specified size region.	LCTMINPF
4	Initiator suppresses CANCEL command except during allocation processing.	LCTCANF
5	Initiator processes only one job.	LCTONEJF
6	Initiator does not process initiator commands.	LCTCMDF
7	Reserved.	

Initiator Options Byte 2: When set on, the bits mean:

Bit	Meaning	Flagname
0	Initiator does not time this job.	LCTTIMEF
1	Initiator does not use checkpoint/restart.	LCTCRF
2	Initiator suppresses DSO processing.	LCTDSOF
3	Initiator operates in hierarchy zero.	LCTINTH0
4	Initiator operates in hierarchy one.	LCTINTH1
5	Initiator uses track stacking.	LCTTSTKF
6	Initiator bypasses allocation recovery.	LCTNORCF
7	Initiator does not wait for data sets.	LCTENQUF

Initiator Options Byte 3: When set on, the bits mean:

Bit	Meaning	Flagname
0-7	Reserved.	

Mapping Macro Instruction: IEFALLCT

Offset Hex	Offset Dec	Field	Length	
0	0	Pointer to First CSCB (or Zero)	4	
4	4	Pointer to GCB Queue	4	
8	8	Master Scheduler ECB	4	
C	12	Master Scheduler IPL Routines ECB	4	
10	16	Pointer to Work Queue UCB	4	
14	20	Pointer to Procedure Library UCB	4	
18	24	Queue Formatting Switch	1	
		Pointer to SET AUTO Command Param. List	3	
1C	28	Pointer to System Log Control Table	4	
20	32	Status Flags 1	1	
		Number of Tracks in Stack	1	
		Interpreter Counter	2	
24	36	Initiator Protection Key Mask	2	
		Minimum Initiator Region Size	2	
28	40	Minimum Job Step Region Size	2	
		Log Status Flags	1	
		Status Flags 2	1	
2C	44	Log ECB	4	
30	48	Pointer to First DSO Control Block	4	
34	52	TSO Monitor Command Flags	1	
		Reserved	1	
		Maximum No. of Broadcast Messages	2	
38	56	Reserved	80	
88	136	Resident Switches	1	Master Common Area
		System Exclusive Flags	1	
		Pending Flags	1	
		ECB Flags	1	
8C	140	Status Flags 3	1	
		Fetch Flags	1	
		Command Verb	8	
90	144			
94	148	Variable Communication Field	8	
98	152			
9C	156	Message Generation Control	2	
A0	160	P Pointer - Points to Character Before List	4	
A4	164	Master ECB	4	
A8	168	Address of ECB in Selected Job Queue Entry of Job Using Console	4	
AC	172	ECB for Allocation Internal Use	4	
B0	176	Pointer to Primary UCB	4	
B4	180	Pointer to Alternate UCB	4	
B8	184	V-type Address of Pseudo-Disable Switch	4	
BC	188	Reserved	4	
C0	192	Reserved	4	

Figure 121. Master Scheduler Resident Data Area

MASTER SCHEDULER RESIDENT DATA AREA

Description: The master scheduler resident data area, which is in the nucleus area of main storage, contains information used by the queue initialization, command scheduling, initiator, and I/O Device Allocation routines. Its location is stored in the CVTMSER and the CVTMSLT fields of the communication vector table.

Most of the fields in the master scheduler resident data area are self explanatory; those fields that require further explanation are described below:

- Queue Formatting Switch: If the high-order bit of this field is on, it indicates that the queue data set must be formatted.

- Status Flags 1: When set on, status flags indicate:

Bit	Definition	Flag Name
0	System Initialization in progress	BAIN
1	MONITOR JOBNAMES	BAJN
2	DISPLAY ACTIVE waiting	BADAW
3	VARY/UNLOAD summary	BAVU
4	Queue hold-release	BAHR
5	DISPLAY ACTIVE processing	BADSP
6-7	Reserved	

- Log Status Flags: When set on, log status flags indicate:

Bit	Meaning
0	Log data set SYSOUT scheduling
1	Log not supported
2	Communications task to stop issuing WTLs
3	Reentry of log writer routine from Damage Assessment routines (DAR)

- Status Flags 2: When set on, the status flags indicate:

Bit	Meaning
0	Master scheduler initialization is complete
1-7	Unused

- TSO MONITOR Command Flags: When set on, flags indicate:

Bit	Meaning	Flag Name
0	MONITOR JOBNAMES issued from TSO terminal	BAMJN
1	MONITOR DSNAME issued from TSO terminal	BAMDSN
2	MONITOR SPACE issued from TSO terminal	BAMSPACE
3	MONITOR STATUS issued from TSO terminal	BAMSTAT
4	MONITOR SESS issued from TSO terminal	BAMSESST
5	MONITOR SESS issued from console	BAMSESSC
6-7	Reserved	

- Resident Switches: When set on, flags indicate:

Bit	Definition	Flag Name
0	IPL switch	MSNIPL
1	SYSOUT IPL	
2	SYSOUT job start	
3-4	Reserved	
5	34 Security	MSCURE34
6	Queue initialized	MSQNIP
7	Procedure catalog initialized	MSPNIP

- System Exclusive Flags: When set on, flags indicate:

Bit	Definition	Flag Name
0	Console Flag	MSCONFLG
1	Cancel flag for ABENDT	MSCANFLG
2	Rollout Flag	MSROLFLG
3	Spinoff flag (CANCEL)	MSSO
4	MONITOR Dataset name	MSSSDSN
5	MONITOR SPACE	MSSSPACE
6		
7		

- Pending flags: When set on, flags indicate:

Bit	Definition	Flag Name
0	IPL Date	MSDATE
1	Region busy	MSPNB
2	Command move completed	MSCMC
3	Interpreter command return	MSICR
4	System input control purge request	MSSYS
5	System output control purge request	MSSYT
6	Blank start pending (REQ=1,START BLANK=0)	MSBSP
7	Console command suppressed by WTO/WTOR Exit Routine	MSCCS

- ECB Flags: When set on, flags indicate:

Bit	Definition	Flag Name
0	External interrupt	MSEXT
1	WTO or WTOR	MSWTO
2	WTL	MSWTL
3	Console Attention key hit	MSATTN
4	System Input	MSYSIN
5	System Output	MSYSOUT
6	Master command routine	MSMCR
7	Summary bit, VARY UCB scan required	MSSUM

- Status Flags 3: When set on, switches indicated:

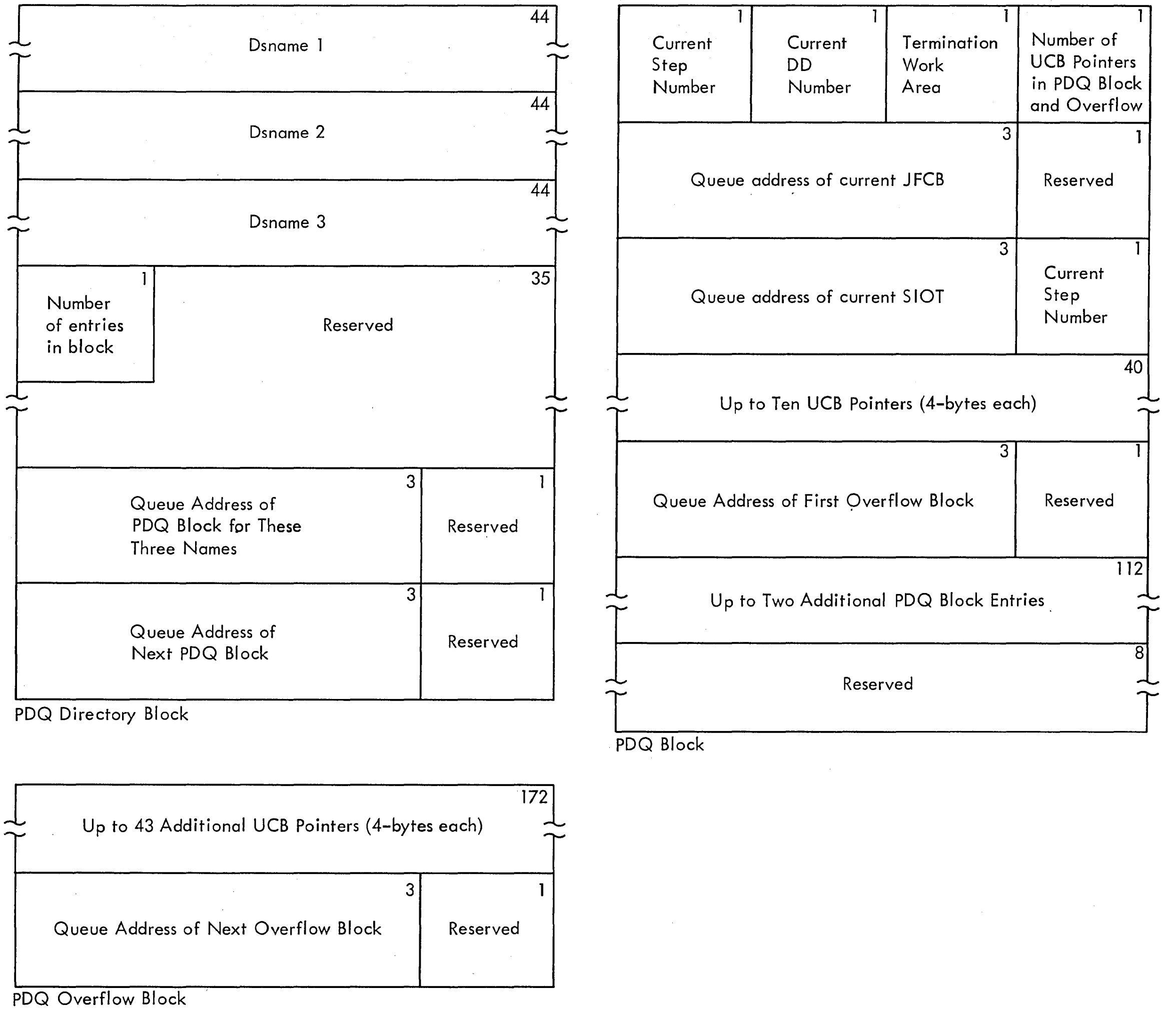
Bit	Definition	Flag Name
0	Master Initialization (IPL) completed	MSSSSIPL or MSINLSW
1	WTO or WTOR pending	MSWRPEN
2	Console usage, Primary or alternate	MSNUPSW
3	Log purge request	MSWRLOG
4	Reader has reached end of file, or	MSREOF
x	Start reader	MSSRDR
5	New reader pending	MSNRP
6	New writer pending	MSYOUT or MSNWP
7	Job notification (1=yes)	MSJNF

- Fetch Flags: When set on, flags indicate:

Bit	Definition	Flag Name
0	Named Fetch	MSNMF
1	Defer current command execution sequence	MSCSD
2	TCB Tree trace Fetch (Locate)	MSTTT
3	Auxilary FETCH given	MSFAX
4	Reply bit to Request attention	MSREPLYB
5	Pseudo-SYSOUT flag	MSPSDT
6	MONITOR Status	MSDISPST
7	Hold-release	MSQHR

Mapping Macro Instruction: IEEBASEA

NOTES


44
Dsname 1
44
Dsname 2
44
Dsname 3
1
Number of entries in block
35
Reserved
3
Queue Address of PDQ Block for These Three Names
1
Reserved
3
Queue Address of Next PDQ Block
1
Reserved
1
Current Step Number
1
Current DD Number
1
Termination Work Area
1
Number of UCB Pointers in PDQ Block and Overflow
3
Queue address of current JFCB
1
Reserved
3
Queue address of current SIOT
1
Current Step Number
40
Up to Ten UCB Pointers (4-bytes each)
3
Queue Address of First Overflow Block
1
Reserved
112
Up to Two Additional PDQ Block Entries
8
Reserved
172
Up to 43 Additional UCB Pointers (4-bytes each)
3
Queue Address of Next Overflow Block
1
Reserved

PDQ Directory Block

PDQ Block

PDQ Overflow Block

Figure 122. Passed Data Set Queue Tables and PDQ Overflow Block

PASSED DATA SET QUEUE

Description: The passed data set queue (PDQ), contains information regarding previously processed data sets which have been passed from executed steps of the job that may be referred to by subsequent steps of the same job. Each PDQ is located in subpool zero of the input queue; it contains a set of tables, consisting of three types of blocks: the PDQ directory block, the PDQ block, and the PDQ overflow block (if required). The PDQ directory block and the PDQ block are created by the JFCB Housekeeping routine. The directory blocks are chained together with pointers, and each PDQ directory block also points to its respective PDQ block. If more than ten additional UCB pointers are needed for any one PDQ entry, one or more PDQ overflow blocks are added in a chain to each such PDQ block entry by Allocation routines.

Initiator routines use the PDQ to obtain pointers to UCBs when allocating devices to passed data sets. Step Termination routines use the PDQ to obtain UCB allocation pointers and data set disposition information.

When control passes to the initiator, the JFCB Housekeeping routine inspects the disposition field of the SIOT for the disposition "PASS" to determine whether a new entry may be required in the PDQ.

If a PASS disposition is found and the dsname is not in the PDQ directory because it was not placed into the directory by a prior PASS, an entry is made in the PDQ for this dsname. If the last PDQ directory block and PDQ block already contain the maximum number of three entries, auxiliary storage space is assigned for a new PDQ directory block and a new PDQ block, thereby providing space for three more dsname entries.

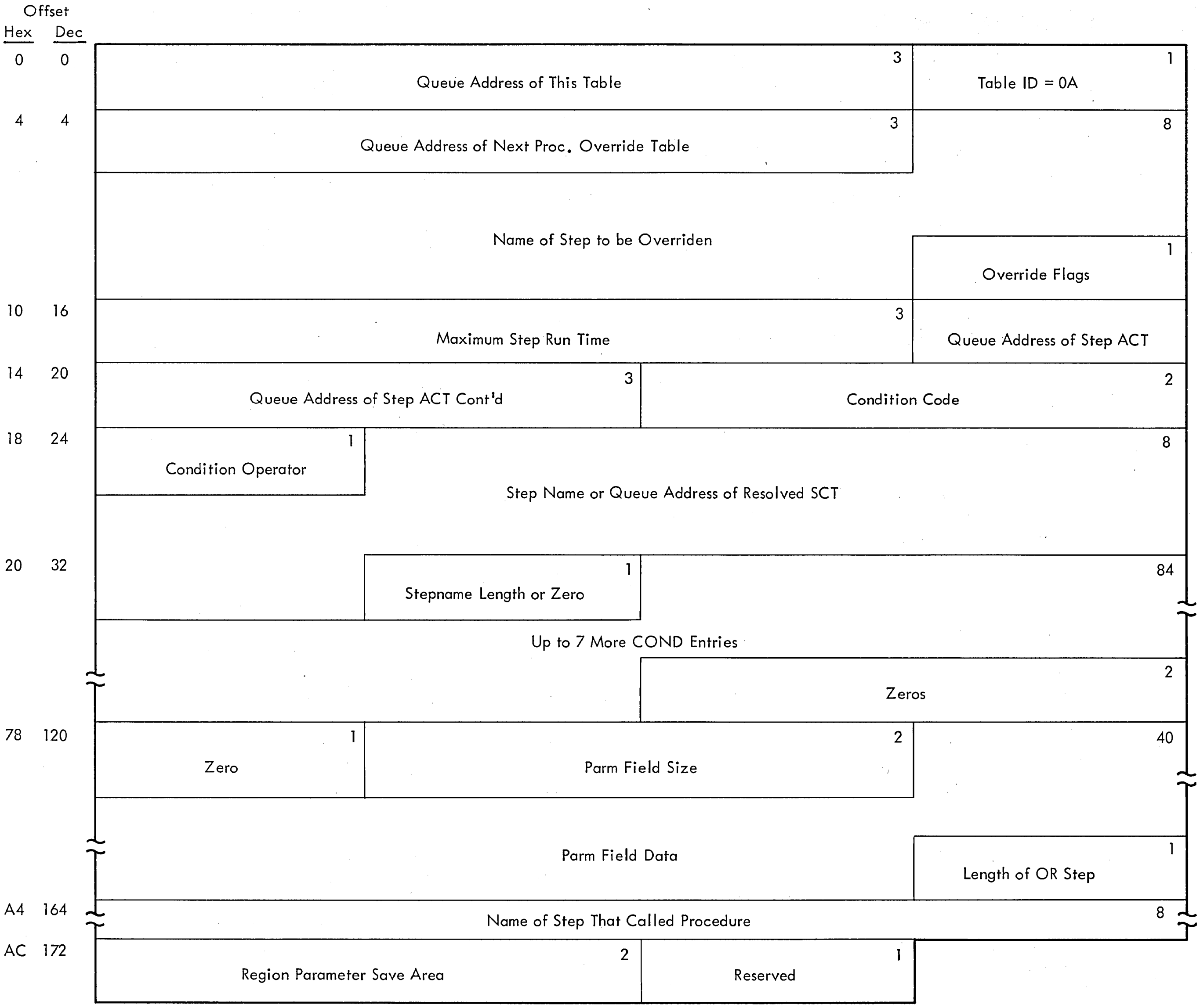
When a passed data set is to be referred to by a subsequent step in the same job, the dsname is specified in the DD statement. The JFCB housekeeping routine checks for the dsname in the PDQ directory to see if the data set was received (passed from a previous step).

If the dsname is found in the PDQ directory, the existing PDQ entry for this dsname is updated to identify the reference as the latest reference to this dsname and the data set is marked as being received in the PDQ entry. If no entry is found, the data set must have been cataloged, so the JFCB routine searches the catalog for this dsname, assuming that this is an initial reference for this job to a cataloged data set.

Bits of the terminate work area byte of the PDQ block have the following status significance:

Bit	Significance	Status
0	Initial status	1 = old
1	Current status	1 = old
2	Pass satisfied	1 = passed 0 = received
3	SYSIN specified	1 = SYSIN
4	SYSOUT specified	1 = SYSOUT

Mapping Macro Instruction: IEFPDQBK



Offset Hex	Offset Dec		
0	0	Queue Address of This Table Table ID = 0A	3 1
4	4	Queue Address of Next Proc. Override Table Name of Step to be Overriden Override Flags	3 8 1
10	16	Maximum Step Run Time Queue Address of Step ACT	3
14	20	Queue Address of Step ACT Cont'd Condition Code	3 2
18	24	Condition Operator Step Name or Queue Address of Resolved SCT	1 8
20	32	Stepname Length or Zero Up to 7 More COND Entries Zeros	1 84 2
78	120	Zero Parm Field Size Parm Field Data Length of OR Step	1 2 40 1
A4	164	Name of Step That Called Procedure	8
AC	172	Region Parameter Save Area Reserved	2 1

Figure 123. Procedure Override Table

PROCEDURE OVERRIDE TABLE

Description: The procedure override table is constructed in the localwork area by the EXEC Statement Processor routine of the interpreter. When the routine is processing an EXEC PROC statement, it constructs a procedure override table for each step to be overridden (specified with a parameter of the form: keyword.STEP=). When a table has been constructed, it is written into the queue data set; when the EXEC PGM statement for the overridden step is processed, the appropriate table is read back into the local work area. Fields from the table are then moved into the SCT or used to resolve condition code referback statements.

Mapping Macro Instruction: IEFVORWA

Offset Hex	Dec				
0	0	3 Queue Address of SCT	1 Table ID (02)	1 Internal Step Status Indicators One	3 Maximum Step Running Time
8	8	2 PARM Count or Step Status Code at Termination	2 Length of Allocate Work Area, or Number of SIOTs	3 Queue Address of First SIOT Entry	1 Reserved
10	16	3 Queue Address of Allocate Work Area	1 Reserved	3 Queue Address of Next SCT	1 Reserved
18	24	3 Queue Address of First SMB for Next Step	1 Reserved	3 Queue Address of Last SMB for this Step	1 Reserved
20	32	3 Queue Address of First ACT Entry for this Step	1 Reserved	3 Queue Address of VOLT	1 Reserved
28	40	3 Queue Address of DSNAME Table for this Step	1 Reserved	Name of Step that Called Procedure	
30	48	8 Name of Step that Called Procedure (continued)		Step Name	
38	56	8 Step Name (continued)		2 Relative Pointer to Step Entry in ACT	2 Length of VOLT
40	64	1 Number of SIOTs in this Step	1 Number of Setup Messages	1 Number of JFCBs to Allocate	1 Step Type Indicators	4 Queue Address of SCTX
48	72	1 X'00'	3 Hierarchy 0 Region Address	1 X'01'	3 Hierarchy 1 Region Address
50	80	3 Address of First Write-to-programmer SMB	1 Count of WTP SMBs for Step	4 Reserved	
58	88	2 Hierarchy 0 Region Size	2 Hierarchy 1 Region Size	2 Reserved	2 Step Dispatching Priority
60	96	4 Step SYSIN Count for SMF		3 Queue Address of PGM=* .stepname.ddname SIOT	
68	104	1 Internal Step Status Indicators Two	3 Queue Address of the Step TIOT	Program Name	
70	112	8 Program Name (continued)		2 Length (in Bytes) of DSNAME Table for this Step	2 First Step Condition Code
78	120	1 First Step Condition Operator	3 Queue Address of First Condition SCT	36	
80	128	Second through Seventh Step Condition Entries			
A0	160	6 Eighth Step Condition Entry or Execute Step after ABEND Information		2 Reserved	
A8	168	3 Queue Address of the First DSB in the Message Class	1 Number of Message Class DSBs for this Step	1 Step Status	3 Queue Address of the Last Legitimate SMB

Figure 124. Step Control Table (SCT)

STEP CONTROL TABLE (SCT)

Description: The step control table (SCT), is used to pass control information to the DD routine of the Interpreter and to the Initiator routines, which also contribute information to the table. This table is created and initialized by the Execute Statement Processor routine of the Interpreter when an EXEC statement is read. One SCT is created for each step of a job.

The following variable-content and indicator fields are included in the table:

Internal Step Status Indicators One:

Bits 0 and 1 pertain to Rollout/Rollin; they correspond to the ROLL parameter on the JOB and EXEC statements:

If bit 0=0, the job cannot be rolled out.
If bit 0=1, the job can be rolled out.
If bit 1=0, the job cannot cause rollout.
If bit 1=1, the job can cause rollout.

Bits 2, 3, and 4 pertain to Checkpoint/Restart; they correspond to the RD parameter in the JOB or EXEC statement:

If bit 2=1, then RD=NR was specified.
If bit 3=1, then RD=RNC or RD=NC was specified.
If bit 4=1, then RD=R or RD=RNC was specified.

Bit 5 is set to 1 by the Reinterpretation Control routine or by the interpreter to indicate that a protection key of zero is to be used for this step.

Bit 7 is set to 1 if an error condition caused the step to be terminated.

Internal Step Status Indicators Two:

Bits 0 and 3 are not used.

If bit 1=1, direct SYSOUT facilities are required to issue separator records and system messages.

If bit 2=1, the Mount Control Volume routine has called for a dummy allocation. Upon completion of the allocation, control is returned to IEFMCVOL.

If bit 4=1, a STEPLIB DD statement is present.
If bit 5=1, SYSIN exists for the job step.
If bit 6=1, the job has ended.
If bit 7=1, the GDG Bias Count table must be updated.

PARM Count or Step Status Code:

a. Interpreter: The number of characters specified in the PARM parameter of the EXEC statement is placed in this field.
b. Initiator: This table entry contains the condition code returned by the processing program.

Step Type Indicators:

Bit 0 is set to 1 if the following parameter definition appears in the EXEC statement:

PGM=*.stepname.ddname

Bit 1 is set to 1 if SYSIN is specified (DD*).

Bit 2 is set to 1 if the parameter associated with a SYSOUT keyword specifies the message class.

Bit 3 is set to 1 if JFCB housekeeping is complete.

Bits 4-6 are used by the initiator as follows:

Bit 4	Bit 5	Bit 6	Action
0	0	0	Use action code
0	0	1	Go to AVR Routine
0	1	0	Go to Space Request Routine
0	1	1	Go to External Action Setup Routine
1	0	0	Go to External Action Verify Routine
1	0	1	Unused
1	1	0	Unused
1	1	1	Unused

Bit 7 is reserved.

Step Code:

Bits 0 and 1 are reserved.

Bit 2 is set to 1 by the initiator to indicate that this SCT corresponds to the first step of the job to be run.

Bits 3-7 are reserved.

Eighth Step Condition Entry: If the EXEC statement COND field specifies eight condition codes, this field contains the eighth entry. If the COND field specifies ONLY or EVEN, or if the step is bypassed or abnormally terminated, the bits of the first byte are set as shown below and the remainder of the eighth step condition entry is set to zero.

Bits 0-2: Reserved

Bit 3: Set on in the EXEC Statement Conditional Execution routine if the step is bypassed because of one or more previous abnormal terminations.

Bit 4: Set on in the EXEC Statement Conditional Execution routine if the step is bypassed because COND=ONLY and no previous abnormal terminations occurred.

Bit 5: Set on in the Step Termination Control routine if the step terminated abnormally.

Bit 6: Set on in the Interpreter if COND=EVEN.

Bit 7: Set on in the Interpreter if COND=ONLY.

Mapping Macro Instruction: IEFASCTB

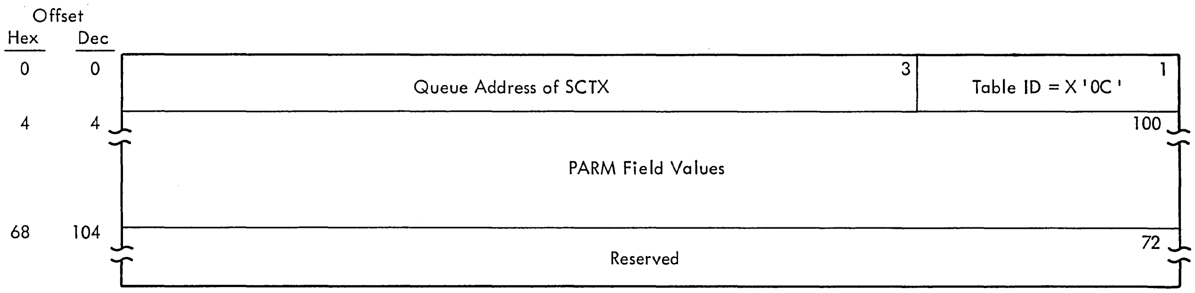


Figure 125. Step Control Table Extension (SCTX)

STEP CONTROL TABLE EXTENSION BLOCK (SCTX)

Description: The step control table extension block (SCTX) contains the information specified in the EXEC statement PARM field. The SCTX is constructed by the EXEC Statement Processor routine of the interpreter in the local word area and written to the job's queue entry only when PARM field values are specified.

Mapping Macro Instruction: None.

Offset Hex	Offset Dec	Field	Length
0	0	Queue Address of SIOT	3
		Table ID = 3	1
4	4	DD Name	8
C	12	Internal DD Numbers of Channel Separation and Affinity Requests	8
14	20	Internal DD Numbers of Unit Separation and Affinity Requests	8
1C	28	Queue Address of Next SIOT	4
		Queue Address of JFCB	4
24	36	Queue Address of SIOT for VOLREF or SUBALLOC	4
		Queue Address of SIOT System/Output Dependency Block	4
2C	44	Reserved	3
		TSO and TCAM Indicators	1
		Reserved	1
		Number of Volumes in VOLT	1
		Relative Pointer to Volume Table Entry	2
34	52	Internal DD Number	1
		Number of Units for this Data Set	1
		Volume Count	1
		Disposition Switches	1
		Indicator Bytes	4
3C	60	Unit Type *	8
44	68	System Output Program Name	8
4C	76	System Output Form Number	4
		System Output Class	1
		DD Statement Duplicate Number	1
		Reserved	2
54	84	TTR of DSB if Sysout Specified	3
		Reserved	1
		Queue Address of Next DSB	4
5C	92	Conditional Disposition	1
		Queue Address of Last SIOT to Pass Data Set	3
		Reserved	26
64	100	Name From DS Name = , Dedicated Work Files	
7C	124	Name From DS Name = , Dedicated Work Files Cont'd	8
		DCB Reference Name	44
84	132	Reserved	2

*If a unit address is specified on a DD statement, the first 6 bytes of this field contain the device type and the last 2 bytes contain the UCB address of the requested unit.

Figure 126. Step Input/Output Table (SIOT)

STEP INPUT/OUTPUT TABLE (SIOT)

Description: The Step Input/Output Table (SIOT), makes DD statement information available to the initiator for use as a source of information for the TIOT and for providing DD information to Allocation and Disposition routines. When a DD statement is read, the interpreter creates a new SIOT and places the DD information into it. The individual bits of the disposition byte and of indicator bytes 57 through 60 and 93 in the SIOT are set to one to indicate the following conditions:

BYTE 47: TSO and TCAM Indicators

Bit 0	TSO DD DYNAM parameter
Bit 1	TSO terminal bit
Bit 1	TSO; 1 = last message class SYSOUT SIOT
Bit 3-6	Not used
Bit 7	TCAM; 1 = QNAME on DD statement

BYTE 55: Scheduler Data Set Disposition Switches

Bit 0	Reserved
Bit 1	Retain volume
Bit 2	Private volume
Bit 3	Pass data set
Bit 4	Keep data set
Bit 5	Delete data set
Bit 6	Catalog data set
Bit 7	Uncatalog data set

BYTE 56: Indicator Byte Number 1

Bit 0	Dummy data set
Bit 1	SYSIN data set
Bit 2	Split (primary)
Bit 3	Split (secondary)
Bit 4	Suballocate
Bit 5	Parallel mount
Bit 6	Unit affinity
Bit 7	Unit separation

BYTE 57: Indicator Byte Number 2

Bit 0	Channel affinity
Bit 1	Channel separation
Bit 2	Volume affinity
Bit 3	JOBLIB DD statement
Bit 4	Unlabeled (no labels)
Bit 5	Nonstandard label
Bit 6	Defer mounting
Bit 7	Received data set

BYTE 58: Indicator Byte Number 3

Bit 0	Volume reference to a DSNAME
Bit 1	SYSIN expected (procedures only)
Bit 2	No associated volume serial in volume table
Bit 3	Volume reference in step
Bit 4	SYSOUT was specified
Bit 5	NEW data set
Bit 6	MOD data set
Bit 7	OLD or SHR data set

BYTE 59: Indicator Byte Number 4

Bit 0	Set by interpreter to indicate GDG single
Bit 1	Set by initiator to indicate GDG all
Bit 2	No PDQ Processing
Bit 3	American National Standard tape label
Bit 4	Step processed
Bit 5	Intra-step volume affinity
Bit 6	Data set is in PDQ
Bit 7	1 = old or modified data set 0 = new data set

BYTE 92: Conditional Disposition

Bits 0-2	Reserved
Bit 3	DD is not private
Bit 4	Keep data set
Bit 5	Delete data set
Bit 6	Catalog data set
Bit 7	Uncatalog data set

Mapping Macro Instruction: IEFASIOT

Offset Hex	Offset Dec					
0	0	Options Selected at IPL 1	Miscellaneous Indicators 1	Offset of SMF Entries in M/S TIOT 2	Pointer to M/S TIOT 4	
8	8	Job Wait Time Limit 4	1/2 SMF Buffer Size 4			
10	16	System ID 2	CPU Model 2	Pointer to Start of SMF Buffer 4		
18	24	Volume Serial 6	Device Status 1	Device Address		
20	32	Device Address (cont'd) 3	Label Status 1	'X' or 'Y' 1	Pointer to Prime DCB 4	
28	40	Volume Serial 6	Device Status 1	Device Address		
30	48	Device Address (cont'd) 3	Label Status 1	'X' or 'Y' 1	Pointer to Alternate DCB 4	
38	56	Write Request ECB 4	Buffer ECB 4			
40	64	Number of Record Segments Required by Logical Record 4	Number of Record Segments Allowed in Data Set 4			
48	72	10-Minute System Wait Time Fields 8				
50	80	'X' or 'Y' Not Found 1	Open DS Routine Switch 1	SMF Fore-ground Options* 1	Reserved 1	Optimum Buffer Write Point 4
58	88	Pointer to XCTL Name 4	DCB Pointer (Always Zero) 4			
60	96	XCTL Name 8				
68	104	Switch A 1	Reserved 3	Time Stamp: Both Data Sets Full		
70	112	Time Stamp: Both Data Sets Full (cont.) 8	Count of Lost Records 4			
78	120	Reserved 2	Current Task TJID 2			

* Not selected at IPL.

Figure 127. System Management Control Area (SMCA)

SYSTEM MANAGEMENT CONTROL AREA (SMCA)

Description: The system management control area (SMCA) is constructed in main storage obtained from subpool 255 by the SMF Initialization routine (module IEESMFIT). The area is used as a communications and work area by the SMF routines.

Although the names of the fields in the SMCA are in most cases sufficient to describe the contents, the following fields require further explanation:

Options Selected at IPL or for Foreground Option at START or MODIFY TSO Time: The field appears twice, at offsets X'0' and X'52'. The field at X'0' is for background jobs, at X'52' for foreground jobs. When the bits in this field are set to 1, they indicate that the corresponding options were selected at IPL or for foreground option at START or MODIFY TSO time.

Bit	Option
0	Job Accounting
1	Step Accounting
2	Dynamic Exits
3	Data Set Accounting
4	Volume Accounting
5	ESV Accounting
6	Temporary Data Set Scratch Records
7	This bit is set to 0 for background jobs; it is set to 1 for foreground jobs.

Miscellaneous Indicators: Bits 0 and 1 are used together as follows:

Bit 0	Bit 1	Meaning
0	0	No recording
0	1	Write SMF records only
1	0	(Invalid combination)
1	1	Write SMF and user records

Bit 2 is set to 0 if operator intervention has been specified in the SMF SYS1.PARMLIB member (OPI=YES); it is set to 1 if OPI=NO.
Bit 3 is set to 1 initially; it is set to 0 when the primary DCB has been opened for the first time.
Bit 4 is set to 1 for SMF device switching.
Bit 5 is set to 1 to indicate that the SMF writer is using the SMF dump routine.
Bit 6 is set to 0 when the left half of the SMF buffer is in use; it is set to 1 when the right half is in use.
Bit 7 is set to 1 to indicate that an SMF dump is being taken.

Device Status: There is a device status field corresponding to each SMF data set (SYS1.MANX and SYS1.MANY). The bits in the fields, when set to 1, have the following meanings:

Bit	Meaning
0	Data set not available for writing records
1	Reserved
2	Direct access device
3	Reserved for internal indicator use
4-5	Reserved
6	Device address defined
7	Volume serial defined

Label Status: There is a label status field corresponding to each SMF data set. The bits in the field, when set to 1, have the following meanings:

Bit	Meaning
0-4	Reserved
5	Nonstandard label (NSL)
6	Standard label (SL)
7	No label (NL)

'X' or 'Y' Not Found: This field is normally set to zero. When, however, the SMF Open Initializer routine (module IEESMFOI) finds that one of the SMF data sets is not present, it stores the appropriate suffix (X for SYS1.MANX, Y for SYS1.MANY).

Open DS Routine Switch: This switch is tested for nonzero by the SVC 83 Open routine each time it is entered. The switch is set to X '80' by this routine each time it passes control to the SVC 83 Allocation routine.

Switch A: The bits in this field, when set to 1, have the following meanings:

Bit	Meaning
0	Reserved
1	Both data sets are full: do not write
2	Reserved
3	Next allocation must be for direct access
4	Allocation search is by volume serial
5	Reserved
6	Writer is checking available space in the data set
7	Writer is switching data sets only

TJID: This field contains the terminal identification number of the swappable TSO task that issued SVC 83. If a non-foreground job issued the SVC 83, the field contains zeros.

Mapping Macro Instruction: IEESMCA

NOTES

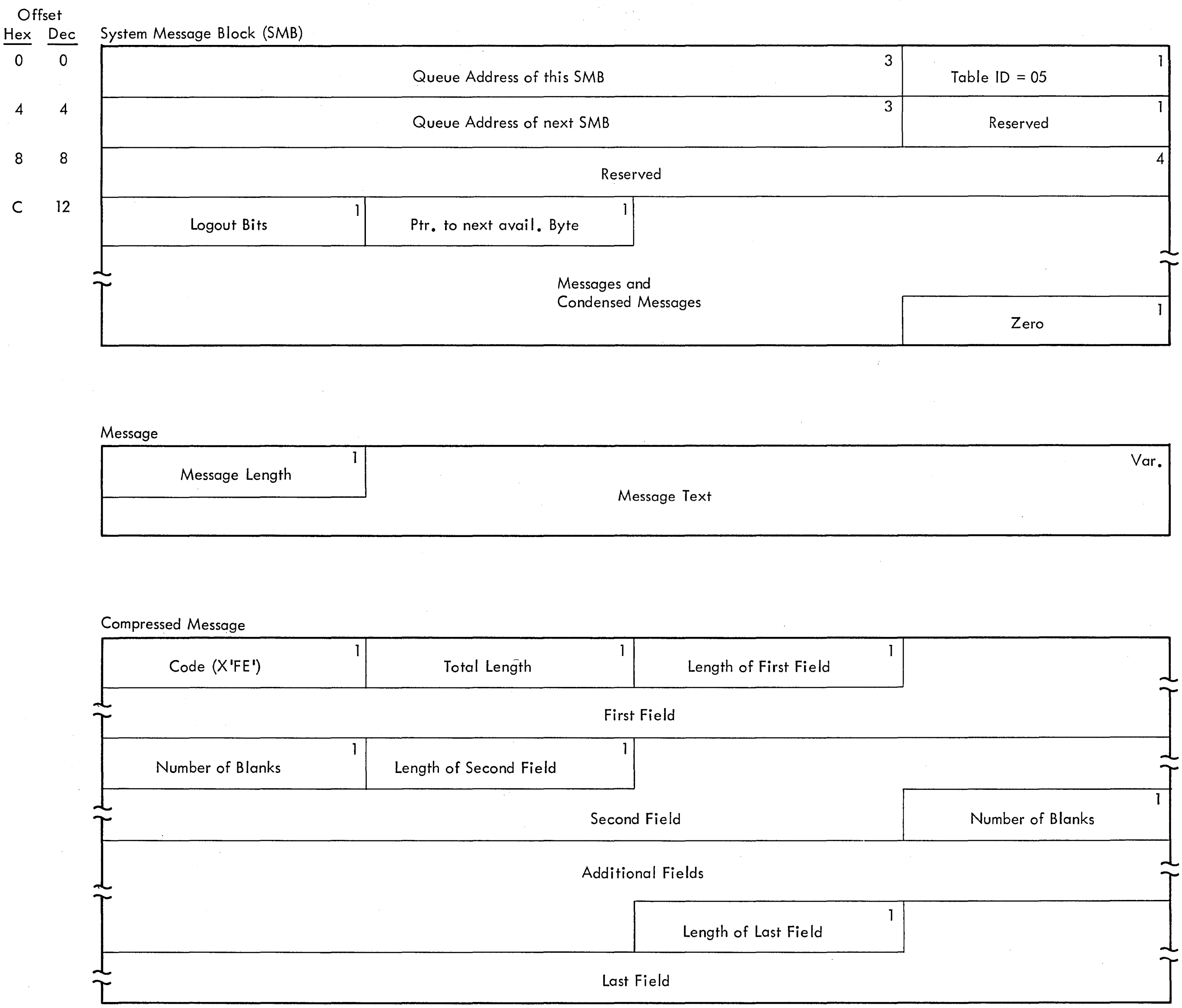


Figure 128. System Message Block (SMB)

SYSTEM MESSAGE BLOCK (SMB)

Description: The system message block (SMB) is used to store messages from the operating system to the programmer until they can be retrieved by a system output writer. The Interpreter Control routine creates one or more SMBs for each job step; the I/O Device Allocation and Termination routines add messages (and if necessary construct additional SMBs). When an SMB has been filled, it is written to the queue data set. When the job is terminated, the job's message class queue entry, which contains the SMBs, is enqueued.

Most of the fields in the SMB are self explanatory; those fields that require further explanation are described below:

- Zero: The last byte of each SMB is set to zero unless the last message completely fills the SMB.
- Message Length: The first byte of each message (except for compressed messages) contains the length of the message.
- Code: If the message is a condensed message (the blanks between the fields have been removed), the first byte of the message contains the hexadecimal characters FE.
- Total Length: This field contains the length of the message, including the code and length fields.
- Number of Blanks: This field indicates the number of blanks that have been removed from between the field preceding it and the field following it. This field is not found following the last message field.

Mapping Macro Instruction: IEFAJCTB

1 Class Name	2 NN of 1st Logical Track Assigned to This Entry	1 No. of Records Assigned in Last Track
1 No. of Logical Trks. Assigned	2 Queue Entry Identification	1 Reserved
3 TTR of Next DSB		1 Zero
4 Pointer to Next SCD Entry in Main Storage		

System Output Class Directory (SCD) Entry: Main Storage Format

1 Class Name	2 NN of 1st Logical Track Assigned to This Entry	1 No. of Records Assigned in Last Track
1 No. of Logical Trks. Assigned	2 Queue Entry Identification	Var
23 Additional Entries		
4 Zeros		
4 TTR of Next 176-Byte SCD (if any)		

System Output Class Directory (SCD): 176-Byte Record Format

Figure 129. System Output Class Directory (SCD)

SYSTEM OUTPUT CLASS DIRECTORY (SCD)

Description: A system output class directory (SCD) is created for each job by the interpreter. As each DD statement specifying a new system output class is encountered, the interpreter creates an entry (using fields from the queue manager parameter area). When the interpreter has completed its processing of the job, it packs the entries into one (or at most, two) 176-byte records, and places the records in the job's input queue entry.

One 176-byte record can contain up to 24 SCD entries; since as many as 36 entries may be required for a job, the additional entries are placed in an additional 176-byte record.

As the steps of the job terminate, the SCD entries are used by the Termination routine to build queue manager parameter areas for writing DSBs to the output queue entries and for enqueuing the entries.

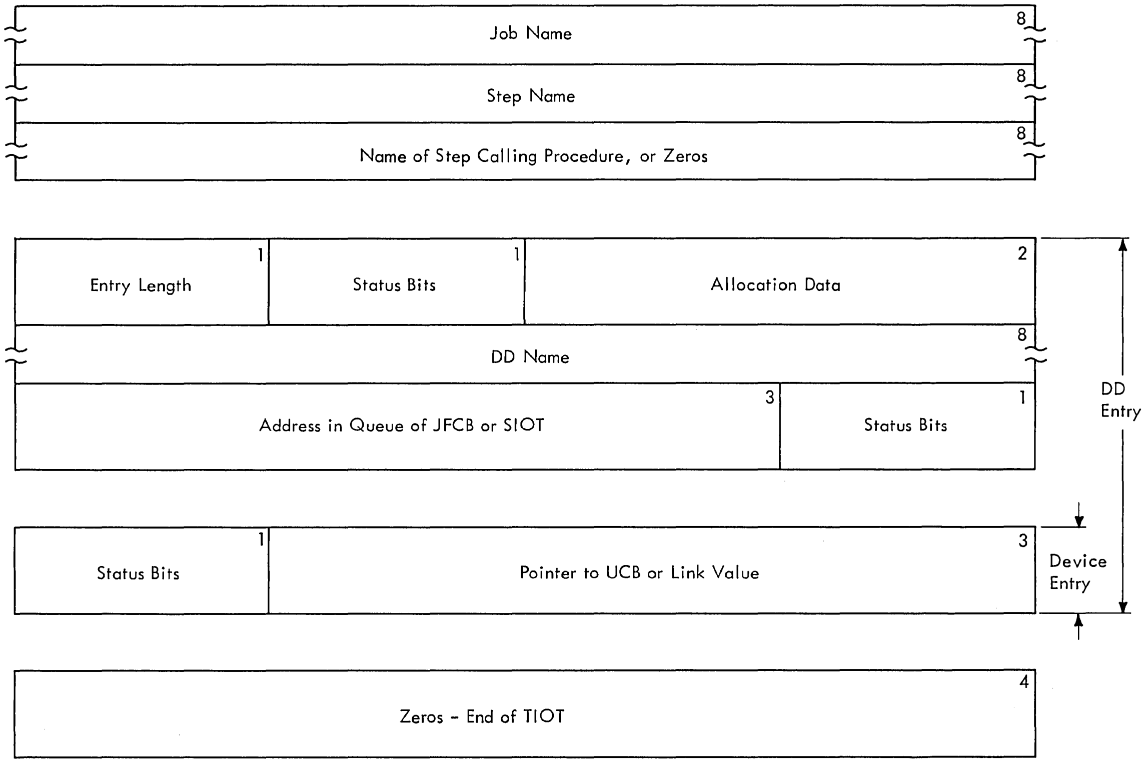


Figure 130. Task Input/Output Table (TIOT)

TASK INPUT/OUTPUT TABLE (TIOT)

Description: The Task Input/output table (TIOT) provides Data Management routines with pointers to the JFCBs and devices allocated to the data sets in a job step or system task. It is constructed in the I/O Device Allocation routine in main storage obtained from subpool zero. When the step is attached, the initiator obtains main storage from subpool 255, and moves the TIOT into that area, where it remains during execution of the step. When the step terminates, the TIOT is moved into main storage obtained from subpool zero during termination processing, then deleted.

For further information on the task input/output table, see the publication IBM System/360 Operating System: System Control Blocks, GC28-6628.

Offset Hex	Offset Dec	Field		Field	
0	0	Queue Address of TCT	3	TCT Switches	1
4	4	Initiator TCB Address	4		
8	8	TCT Core Table Start Address	4		
C	12	TCT I/O Table Start Address	4		
10	16	Subpool	1	Size of TCT	3
14	20	User Time Limit Routine Address	4		
18	24	Pointer to User Parameter List	4		
1C	28	Job Management Record Address	4		
20	32	User Output Limit Routine Address	4		
24	36	Step Task Time Field Overflow	4		
28	40	Step Accumulative Task Time	4		
2C	44	Job/Step Maximum Wait Time	4		
30	48	Number of Input Terminal Lines	4		
34	52	Number of Output Terminal Lines	4		
38	56	High Address Allocated from Bottom of Hierarchy 0 Region	4		
3C	60	Allocation Start Time	4		
40	64	Problem Program Start Time	4		
44	68	Low Address Allocated from Top of Hierarchy 0 Region	4		
48	72	Number of 2K Blocks Allocated	2	Hierarchy 0 Region Request (in 2K Blocks)	2
4C	76	Current Amount of Borrowed Hierarchy 0 Storage (2K Blocks)	2	Maximum Amount of Borrowed Hierarchy 0 Storage (2K Blocks)	2
50	80	High Address Allocated from Bottom of Hierarchy 1 Region	4		
54	84	Low Address Allocated from Top of Hierarchy 1 Region	4		
58	88	Number of 2K Blocks Allocated	2	Hierarchy 1 Region Request (in 2K Blocks)	2
5C	92	Current Amount of Borrowed Hierarchy 1 Storage (2K Blocks)	2	Maximum Amount of Borrowed Hierarchy 1 Storage (2K Blocks)	2

Figure 131. Timing Control Table (TCT)

TIMING CONTROL TABLE (TCT)

The timing control table (TCT) is constructed in main storage obtained from subpool 253 by the User Exit Initialization routine (module IEFSMFIE) of the initiator. The table is passed to the supervisor when the job step is established; it is an information source for the SMF Termination record, and the main storage it occupies is released when the step is terminated.

For further information on the TCT, see the System Control Blocks publication.

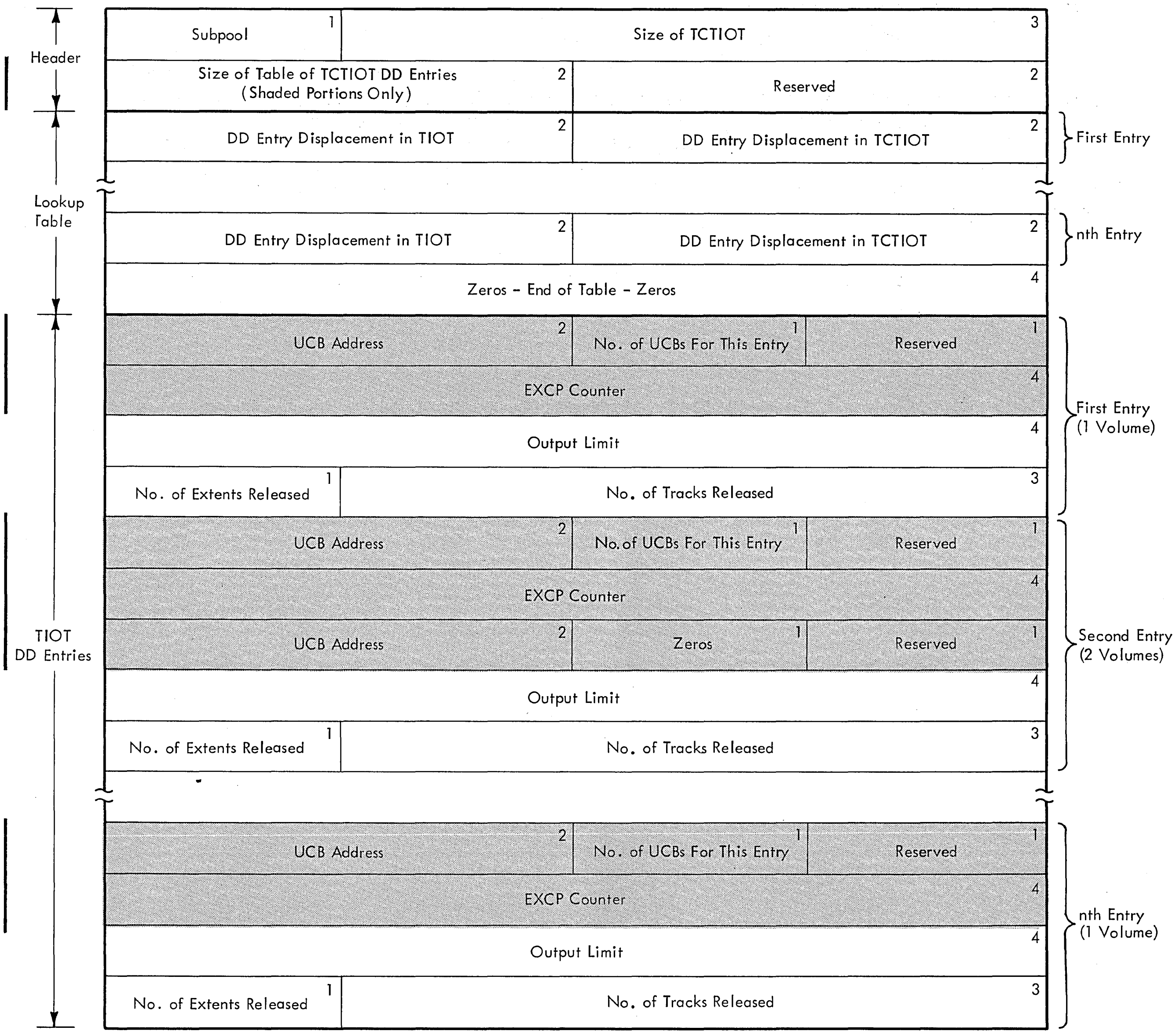


Figure 132. TCT I/O Table (TCTIOT)

TIMING CONTROL TASK INPUT/OUTPUT TABLE (TCTIOT)

The timing control task input/output table (TCTIOT) is constructed in main storage obtained from subpool 253 by the TCTIOT Construction routine (module IEFSMFAT) before the job step task is established. The table is passed to the supervisor, and is used as an information source for the SMF Termination record. The main storage occupied by the TCTIOT is released when the step is terminated.

For further information on the TCTIOT, see the System Control Blocks publication.

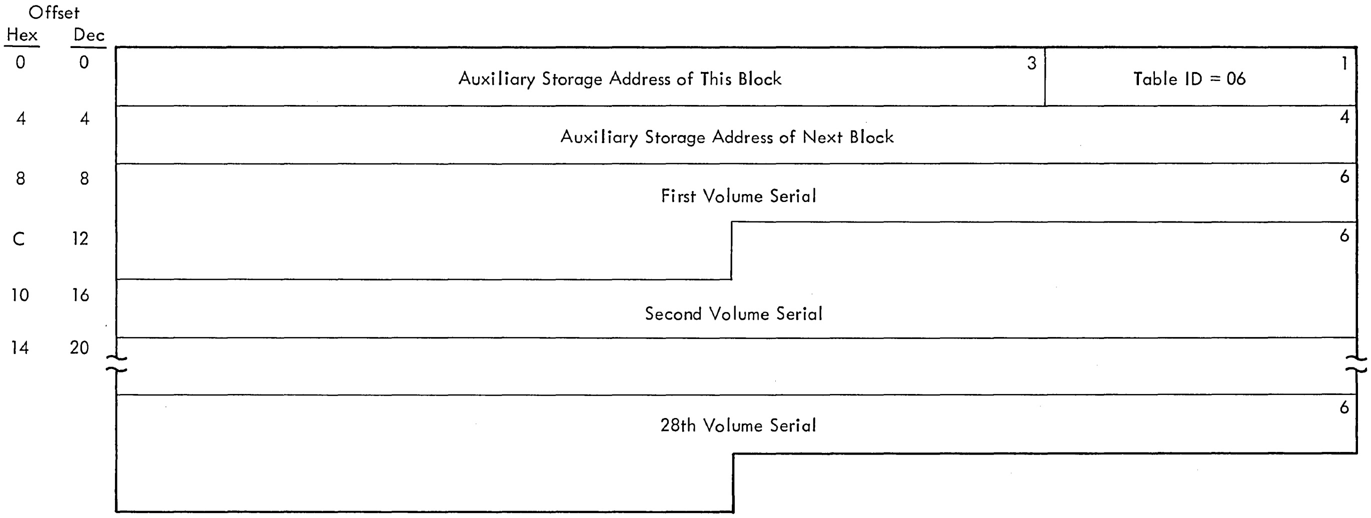


Figure 133. Volume Table (VOLT)

VOLUME TABLE

Description: The volume table (VOLT) consists of a series of chained blocks and contains the list of volume serial numbers to be used in a given step. Use of the list reduces the number of times auxiliary storage must be referenced during allocation. The table is built by the DD routine for each step, and is modified by the JFCB Housekeeping routine. The maximum extent of each block of the table is 176 bytes, and the maximum number of volumes listed per block is 28.

Offset			
Hex	Dec		
0	0	TTR of SMB used for WTP (WTPSMB) 3	WTP Flags (WTPFLGSA) 1
4	4	No. of Bytes Remaining in WTP SMB for WTP Message (WTPBYTES) 1	Address of Message Class QMPA (WTPQMPA) 3
8	8	TTR of First WTP SMB (WTPCRSMB) 3	Number of WTP SMBs Used in Step (WTPCRCNT) 1
C	12	WTP Record Limit Counter (WTPLIMIT) 1	TTR of Reserved WTP SMBs (WTPRSMBS) 3

Figure 134. Write-to-Programmer Control Block (WTPCB)

WRITE-TO-PROGRAMMER CONTROL BLOCK (WTPCB)

Description: The write-to-programmer control block (WTPCB) contains information that is necessary to place WTP messages into the appropriate message class SMB. When a job is ready to be processed, the Job Selection routine (module IEFSD161) creates the WTPCB from subpool 253, initializes it, and chains the WTPCB from the job step control block (JSCB). Following allocation for each job step, the Allocation Exit routine (module IEFSD41Q) initializes the WTPCB with the WTP SMB address, the SYSOUT indicators, and the address of the QMPA for the message class data set. The Step Delete routine (module IEFSD164) reinitializes the WTPCB at step termination for use by the next step; at job termination, the routine reinitializes the WTPCB to zeroes for use by the next job.

The fields in the WTPCB are described below:

- WTPSMB: This field contains the TTR of the SYS1.SYSJOBQE record used by the WTP SMB.

- WTPFLGSA: The bit setting in this field indicates the type of WTP processing being performed. The bits, when set to one, have the following meaning:

Bit	Meaning
0	Error or problem with job queue I/O.
1	Limit message has been processed.
2	Step already contains SYSOUT; therefore, WTPSMB field contains the TTR of the SMB assigned for termination.
3	Error caused by system task; therefore, return from WTP Error Processing routine (module IEFWTP02) to WTP Message Processing routine (module IEFWTP01) to process WTP message.
4	"No record" message processed, indicating no SMBs available for messages.
5	Last SMB for job has been used.
6	WTP has been invoked for this step.
7	Additional routing code; therefore, return to WTOR or WTO.

- WTPBYTES: This field indicates the number of bytes that are remaining in the WTP SMB for a WTP message.

- WTPCRSMB: This field contains the address of the first WTP SMB. This address is needed for checkpoint/restart processing.

- WTPCRCNT: This field contains a counter that indicates the number of WTP SMBs used by the step.

- WTPLIMIT: This field contains the limit number of WTP records.

- WTPRSMBS: This field contains the address of the reserved SMBs used for WTP.

Mapping Macro Instruction: IEFWTPCB

Appendix B: Module Descriptions

This appendix contains a brief description of each of the modules required at the MVT level of job management. Modules are listed alphabetically, by module name (as it appears in the MODLIB data set); associated with each module name is a descriptive name, which indicates the major component of the system to which the module belongs. Each module description contains a brief statement of the purpose of the module. Where applicable, the description includes the names of the module's entry points, the (MODLIB) names of the modules to which it passes control (exits), the major tables and work areas to which it refers, its attributes, and the names of the control sections it contains.

IEEBASEA: Master Scheduler Resident Data Area

The purpose of this module is to contain information used by Job Management routines. It is described in the Tables and Work Areas section of this publication.

IEECVINT: See the MVT Supervisor PLM

IEELWAIT: See the MVT Supervisor PLM

IEEMPCKR: SVC 34 - VARY Operand Check Routine

The purpose of this module is to check the syntax of the VARY CPU, VARY channel, and VARY storage commands.

- Entry: IGC2203D
- Exit: XCTL to IGC0503D or IGC2103D or IGC0803D
- Attributes: Disabled, locked, reenterable
- Control Sections: IGC2203D

IEEMPVCH: VARY Channel Routine

The purpose of this module is to logically reconfigure channels online or offline.

- Entry: IEEMPVCH
- Exit: Branch on register 14
- Attributes: Disabled, locked, reenterable
- Control Sections: IEEMPVCH

IEEMPVCP: VARY CPU Routine

The purpose of this module is to logically reconfigure CPUs online or offline.

- Entry: IEEMPVCP
- Exit: Branch on register 14
- Attributes: Disabled, locked, reenterable
- Control Sections: IEEMPVCP

IEEMPVSE: VARY Storage Routine

The purpose of this module is to logically reconfigure storage blocks online or offline.

- Entry: IEEMPVSE
- Exit: Branch on register 14
- Attributes: Disabled, locked
- Control Sections: IEEMPVSE

IEEPALTR: Queue Alter Get Region Routine

The purpose of this routine is to obtain a region of main storage for the Queue Alter routine.

- Entry: IEEPALTR
- Exit: To IEESD562, IEE00110 (SVC 110), IEEVATT1, return to caller
- Attributes: Read-only, reenterable
- Control Sections: IEEPSTP1

IEEPDISC: DISPLAY CONSOLES Get Region Routine

The purpose of this routine is to obtain a region of main storage, and set up an environment for the execution of the DISPLAY CONSOLES command, and then free the region when control is returned.

- Entry: IEEPDISC, from IEEVWAIT

- Exit: To IEEXEDNA, via (SVC 3)
- Attributes: Read-only, reenterable , resident
- Control Sections: IEEPDISC

IEEPPRES: Master Scheduler Get Region Routine

The purpose of this routine is to obtain a region of main storage for the routines that set public/private volume attributes.

- Entry: IEEPPRES
- Exit: To IEEVPRES
- Attributes: Reentrant, resident in link-pack area.
- Control Sections: IEEPPRES

IEEPRTN2: Free Region Routine

The purpose of this routine is to free the region obtained for the execution of the START or MOUNT commands.

- Entry: IEEPRTN2
- Exit: Return to caller
- Attributes: Read-only, reenterable
- Control Sections: IEEPRTN2

IEEPRWI2: START and MOUNT Commands Get Region Routine

The purpose of this routine is to obtain a region of main storage for the execution of the START and MOUNT Commands.

- Entry: IEEPRWI2
- Exit: XCTL to IEEVSTAR for START, IEEVMNT1 for MOUNT
- Attributes: Read-only, reenterable
- Control Sections: IEEPRWI2

IEEPSN: System Task Control-Interpreter Post Scan Exit Routine

The purpose of this routine is to verify that the program name in the EXEC statement of a starting procedure is in the table of system tasks that qualify to be started.

- Entry: IEEPSN from IEFVFA
- Exit: Branch on register 14 for return to caller
- Attributes: Reentrant
- Tables/Work Areas: Linkage table
- Control Sections: IEEPSN

IEESD562: Queue Alter Syntax Check Routine

The purpose of this module is to check the syntax of those commands executed by the Queue Alter routine.

- Entry: IEESD562 from IEEPALTR
- Exit: To IEESD582, IEESD565, IEESD566, IEESD563, IEEXEDNA
- Tables/Work Areas: QCR, XSA, CSCB, CVT, QMPA
- Attributes: Reentrant. Character Dependence Type C
- Control Section: IEESD562

IEESD563: Queue Alter Routine - Search Control Routine

The purpose of this routine is to control the search of the work queues during execution of queue manipulation commands.

- Entry: IEESD563, from IEESD582, IEESD564, IEESD583, IEESD578, or IEESD565
- Exit: To IEESD564, IEESD565, IEESD563, IEESD575, or IEESD584, via XCTL
- Attributes: Reentrant
- Tables/Work Areas: QCR, QMPA, CVT, XSA, CSCB
- Control Sections: IEESD563

IEESD564: Queue Alter Routine - Search Routine

The purpose of this routine is to search the work queues as necessary for the execution of the queue manipulation commands.

- Entry: IEESD564, from IEESD563 and IEESD584
- Exit: To IEESD583 via XCTL, IEESD584, IEESD563
- Attributes: Reentrant
- Tables/Work Areas: QCR, XSA, CSCB, CVT, QMPA
- Control Sections: IEESD564

IEESD565: Queue Alter Routine - Service Routine

The purpose of this routine is to cause messages to be written to the operator or TSO user by the master scheduler Message Assembly routine (IEE0503D or IEE2103D) and to use queue management to enqueue job entries as required during execution of the queue manipulation commands.

- Entry: IEESD565, from IEESD582, IEESD583, IEESD562, or IEESD563
- Exit: To IEE0503D, IEE2103D or IEFQMNQQ via LINK; return to IEEPALTR; or XCTL to IEESD563
- Attributes: Reentrant
- Tables/Work Areas: QMPA, CSCB, XSA, QCR, CVT
- Control Sections: IEESD565

IEESD575: Queue Alter - Queue Scratch Routine

The purpose of this routine is to construct the scratch parameter list.

- Entry: IEESD575 from IEESD563, IEESD581 or IEESD583
- Exit: XCTL to IEESD581, IEESD576
- Tables/Work Areas: CSCB, CVT, JCT, JFCB, QMPA, SCT, SIOT, TIOT, UCB, UCB lookup table
- Attributes: Read-only, reenterable
- Control Sections: IEESD575

IEESD576: Queue Alter - Queue Alter Delete Routine

The purpose of this routine is to delete queue entries by invoking the Queue Manager Delete routine (module IEFQDELQ).

- Entry: IEESD576 from IEESD575
- Exit: XCTL to IEESD578
- Tables/Work Areas: CSCB, CVT, DSB, JCT SCD, SMB
- Attributes: Read-only, reenterable
- Control Sections: IEESD576

IEESD577: Queue Alter - Queue Restart Enqueue Routine

The purpose of this routine is to enqueue data sets for a cancelled restarting job by invoking the Queue Manager Enqueue routine (module IEFQMNQQ).

- Entry: IEESD577 from IEESD578
- Exit: XCTL to IEESD579
- Tables/Work Areas: CSCB, QMPA, SCD, SMB
- Attributes: Read-only, reenterable
- Control Sections: IEESD577

IEESD578: Queue Alter - Queue Message Class Setup Routine

The purpose of this routine is to reinitialize the message class DSBs to zeros and to initialize the QMPA for enqueuing the message class.

- Entry: IEESD578 from IEESD576
- Exit: XCTL to IEESD563, IEESD577, IEESD579, or IEESD580
- Tables/Work Areas: CSCB, CVT, JCT, QMPA, SCD, SMB
- Attributes: Read-only, reenterable
- Control Sections: IEESD578

IEESD579: Queue Alter - Queue SMB Routine

The purpose of this routine is to place the appropriate cancel message in the SMB, invoke the Queue Manager Enqueue routine to enqueue the message class, and issue the message to the operator.

- Entry: IEESD579 from IEESD577 or IEESD578
- Exit: Return to IEEPALTR
- Tables/Work Areas: CSCB, QMPA, CVT, SMB
- Control Sections: IEESD579, IEEMSWTO

IEESD580: Queue Alter - Message Routine

This routine issues the cancel message to the operator if there is no message class associated with the job.

- Entry: IEESD580 from IEESD578
- Exit: Return to IEEPALTR

- Tables/Work Areas: CSCB, QMPA

- Control Sections: IEESD580

IEESD581: Queue Alter - Queue Scratch Routine

The purpose of this routine is to issue the SCRATCH macro instruction to scratch data sets.

- Entry: IEESD581 from IEESD575

- Exit: XCTL to IEESD575

- Control Sections: IEESD581

IEESD582: Queue Alter Routine -- ECB/IOB Construction Routine

This routine constructs an ECB/IOB to be used when searching the work queues during execution of the queue manipulation commands.

- Entry: IEESD582, IEESD82A from IEESD562

- Exit: XCTL to IEFSD563 or IEESD565

- Attributes: Reentrant

- Tables/Work Areas: QCR, XSA, CSCB, CVT, QMPA

- Control Section: IEESD582

IEESD583: Queue Alter Routine -- Queue Search Return Routine

This routine determines the work queues that are to be searched during execution of the queue manipulation commands.

- Entry: IEESD583 from IEESD564, IEESD563, or IEESD584

- Exit: XCTL to IEESD563, IEESD565, or IEESD575

- Attributes: Reentrant

- Tables/Work Areas: QCR, CSCB, CVT, QMPA, UCM, Queue Manager DCB

- Control Sections: IEESD583

IEESD584: DISPLAY Q/DISPLAY N Message Setup Routine

This routine issues the WTO or TPUT macro instruction for the DISPLAY Q/DISPLAY N control and label lines.

- Entry: IEESD584 from IEESD563

- Exit: IEESD564, IEESD583

- Attributes: Reentrant

- Tables/Work Areas: CSCB

IEESMFAL: SVC 83 -- SMF Allocation Routine

The purpose of this routine is to allocate devices for the SMF data sets.

- Entry: IEESMFAL

- Exit: To IEESMFOP

- Attributes: Reentrant

- Tables/Work Areas: SMCA, CVT

- Control Sections: IEESMFAL

IEESMFIT: SMF Initialization Routine

The purpose of this routine is to initialize the SMCA, and to read member SMFDEFLT from the SYS1.PARMLIB data set.

- Entry: IEESMFIT, IEESMFI4

- Exit: To IEESMFI3 and IEESMFOI or IEEVWAIT

- Attributes: Non-reenterable

- Tables/Work Areas: CVT, DCB, JFCB, M/S Resident Data Area, SMCA, TIOT, UCB

- Control Sections: IEESMFIT

IEESMFI2: SMF Initialization Routine

The purpose of this routine is to construct the SMF IPL, I/O and direct access device records, and to initialize the 10-minute timer.

- Entry: IEESMFI2

- Exit: To IEEVWAIT

- Attributes: Non-reenterable

- Tables/Work Areas: SMCA, CVT, DCB, JFCB

- Control Sections: IEESMFI2

IEESMFI3: SMF Parameter Processor

The purpose of this routine is to process the SMFDEFLT parameters and the SMF parameters entered from the operator's console.

- Entry: IEESMFI3, IEESMFIO, IEESMFMS

- Exit: To IEESMFIT (entry point IEESMFI4)

- Attributes: Serially Reusable

- Tables/Work Areas: CVT, SMCA, UCB, M/S Resident Data Area

- Control Sections: IEESMFI3

IEESMFOI: SMF Open Initializer

The purpose of this routine is to store the DCBs and JFCBs for the SMF data sets, and to establish the SMF task.

- Entry: IEESMFOI
- Exit: IEESMFI2 or IEEVWAIT
- Attributes: Serially reusable
- Tables/Work Areas: DCB, JFCB, SMCA, TIOT
- Control Sections: IEESMFOI

IEESMFOP: SVC 83 -- Open Routine

The purpose of this routine is to open the SMF data sets, and to switch between the primary and alternate data sets.

- Entry: IEESMFOP
- Exit: IEESMF8C
- Tables/Work Areas: DCB, JFCB, SMCA, TIOT
- Attributes: Reentrant
- Control Sections: IEESMFOP

IEESMFWR: SMF Writer Routine

The purpose of this routine is to write the contents of the SMF buffer in the SMF data set.

- Entry: IEESMFWR
- Exit: None
- Tables/Work Areas: CVT, DCB, SMCA
- Attributes: Reentrant
- Control Section: IEESMFWR

IEESMF8C: SVC 83 -- Record Transfer Routine

The purpose of this routine is to place user records into the SMF buffer, and to cause the records to be written when the buffer is full.

- Entry: IEESMF8C
- Exit: IEESMFOP
- Tables/Work Areas: CVT, SMCA
- Attributes: Reentrant
- Control Sections: IGC0083

IEEVDRGN: Display Region Size Routine

The purpose of this routine is to determine the size of the region occupied by a job step.

- Entry: VDRGN
- Exit: Return to caller
- Tables/Work Areas: CVT, TCB, SPQE, DQE, PQE, FBQE
- Attributes: Supervisor state, non-reenterable, resident, disabled for all maskable interruptions except machine check.
- Control Sections: IEEVDRGN

IEEVDSP1: Display Active Routine

The purpose of this routine is to execute the DISPLAY A command.

- Entry: IEEVDSP1, from IEEVWAIT
- Exit: IEEVDRGN, IEE00110 (via SVC 110) and return
- Tables/Work Areas: CVT, TCB, RB, M/S resident data area, TIOT, CSCB, RCB (TSO), TSCVT (TSO)
- Control Sections: IEEVDSP1

IEEVICLR: Internal JCL Reader Routine

The purpose of this routine is to read the internal job control language used in starting a reader or writer.

- Entry: IEEVICLR, IEFICR
- Exit: Return to caller
- Tables/Work Areas: DCBD
- Attributes: Read-only, reenterable
- Control Sections: IEEVICLR

IEEVICTL: System Task Control - Initiator Control Routine

The purpose of this routine is to provide an interface between the System Task Control routine and the Initiator subroutine.

- Entry: IEEVICTL from IEEVRCTL, IEEVIC upon return from the initiator

- Exit: XCTL to IEFSD060, XCTL to IEEPRTN2, CALL to IEEVSMSG

- Tables/Work Areas: CVT, CSCB, IEL, JSCB, JCT, TIOT, TCB

- Control Sections: IEEVICTL

IEEVIPL: Master Scheduler IPL Routine

The purpose of this routine is to cause the initialization of the communications task and the queue data set, to initialize the master scheduler resident data area, to cause volume attributes to be set for resident volumes, and to process the initial SET command.

- Entry: IEEVIPL

- Exit: To IEEVWAIT, IEESMFIT, IEFQINTZ, IEEPPRES, IEEVLIN, IEECVCTI

- Attributes: Non-reenterable

- Tables/Work Areas: IEEBASEA, CVT, XSA, UCB

- Control Sections: IEEVIPL

IEEVJCL: System Task Control Routine - JCL Edit Routine

The purpose of this routine is to construct the internal job control language used in executing the START and MOUNT commands.

- Entry: IEEVJCL, from IEEVSTAR or IEEVMNT1

- Exit: XCTL to IEEVRCTL

- Tables/Work Areas: SDT, CSCB

- Attributes: Reentrant

- Control Sections: IEEVJCL

IEEVLDSP: See the MVT Supervisor PLM

IEEVLIN1: See the MVT Supervisor PLM

IEEVLNKT: System Task Control Routine - Link-Table Module

This nonexecutable module consists of a table of the program names that are specified in the EXEC statement for system tasks initiated via a START command. The Interpreter Post Scan Exit routine (module IEEPSN) scans the list during its processing.

- Attribute: Refreshable

IEEVLOUT: See the MVT Supervisor PLM

IEEVMNT1: MOUNT Command Syntax Check Routine

The purpose of this routine is to check the syntax of the MOUNT command, and to build a start descriptor table containing the parameters of the command.

- Entry: IEEVMNT1

- Exit: To IEEVJCL or IEE0503D

- Tables/Work Areas: CVT, TIOT, XSA, CSCB, SDT, CIB

- Attributes: Reentrant

- Control Sections: IEEVMNT1

IEEVMNT2: MOUNT Command Control Routine

The purpose of this routine is to check the status of the unit to be mounted, and to set the appropriate bits in the unit control block.

- Entry: IEEVMNT2

- Exit: To IEEVSMSG, IEE0503D, or return to caller

- Table/Work Areas: CSCB, TIOT, CIB, CVT, UCB, XSA

- Attributes: Reentrant

- Control Sections: IEEVMNT2

IEEVOMSG: System Task Control Routine - Message Writing Routine

The purpose of this routine is to assemble and write messages to the operator.

- Entry: IEEVOMSG

- Exit: Return to caller

- Control Sections: IEEVOMSG

IEEVPRES: Master Scheduler Public/Private Interface Routine

The purpose of this routine is to set up the interface for module IEFPRES.

- Entry: IEEVPRES

- Exit: To IEFPRES

- Tables/Work Areas: UCB

- Attributes: Reentrant

- Control Sections: IEEVPRES

IEEVRC: System Task Control Routine - Interpreter Exit Routine

This routine checks the return codes passed by the interpreter and the Post Scan Exit routine and sets the job-failed bit in the JCT if errors are indicated.

- Entry: IEEVRC

- Exit: Branch to IEEVICTL. BALR to IEFCNVRT or IEFRDWRT

- Tables/Work Areas: CVT, CSCB, M/S resident data area, QMPA, NEL, JCT, TIOT, JSWA, JSEL, JSOL, JSXL

- Control Sections: IEEVRC, IEEVICTL, IEFCNVRT, IEFRDWRT, IEEVSMSG

IEEVRCTL: System Task Control Routine - Interpreter Control Routine

The purpose of this routine is to provide an interface between the System Task Control routine and the Interpreter subroutine.

- Entry: IEEVRCTL

- Exit: XCTL to IEFVH1

- Tables/Work Areas: CVT, CSCB, M/S resident data area, QMPA, NEL, JCT, TIOT, JSWA, JSEL, JSOL, JSXL

- Control Sections: IEEVRCTL

- Attribute: Refreshable

IEEVRFRX: See the MVT Supervisor PLM

IEEVRJCL: System Task Control - Internal JCL Reader Routine

The purpose of this routine is to read the internal JCL built by the System Task Control routine for the START and MOUNT commands.

- Entry: IEEVRJCL

- Exit: Branch on register 14 to the interpreter

- Tables/Work Areas: DCB

- Control Sections: IEEVRJCL

IEEVSIPL: SYS1.BRODCAST TIOT Initialization Module

The purpose of this routine is to initialize the DD entry of the Master Scheduler's TIOT for the SYS1.BRODCAST data set.

- Entry: IEEVSIPL

- Exit: To IEEVIPL (this is return to caller)

- Tables/Work Areas: CVT, TCB, TIOT

- Control Sections: IEEVSIPL

- Attributes: Reusable

IEEVSMSG: Message Writer Routine

The purpose of this routine is to write messages to the operator as required by the master scheduling task and the System Task Control routine.

- Entry: IEEVSMSG, from IEEVMNT2, IEEVICTL, IEFVH1 or IEFIRC

- Exit: Return to caller

- Control Sections: IEEVSMSG

IEEVSTAR: System Task Control Routine - START Command Syntax Check Routine

The purpose of this routine is to check the syntax of a START command, and to build a start descriptor table containing the parameters of the command.

- Entry: IEEVSTAR from IEEPRWI2

- Exit: To IEEVJCL or IEE0503D

- Tables/Work Areas: SDT, M/S Resident Data Area, CVT, M/S TIOT, UCB XSA, CSCB, CIB, JSCB

- Attributes: Reentrant

- Control Sections: IEEVSTAR

IEEVWAIT: Master Scheduler Wait and Attach Routine

The purpose of this routine is to wait for a task-creating command or a system log operation, to scan the CSCB chain for elements with the assignment pending bit on, and to attach the appropriate command execution tasks.

- Entry: IEEVWAIT

- Tables/Work Areas: M/S resident data area, UCM, CSCB

- Control Sections: IEEVWAIT

IEEXEDNA: DISPLAY CONSOLES Processor

The purpose of this routine is to process the DISPLAY command with the CONSOLES operand, cause a display of the system console configuration on the requesting console.

- Entry: IEEXEDNA, from IEEPDISC

- Exit: IEEPDISC

- Attributes: Reentrant

- Control Sections: IEEXEDNA

IEE0003D: SVC 34 - STAE Environment Creation Routine

The purpose of this routine is to create a STAE environment for the command scheduler. When a failure occurs during the command scheduling task, this routine passes control to the STAE Exit routine.

- Entry: IEE0003D

- Exits: XCTL to IEE0303D (after STAE environment is created), IEE5103D (when SVC 34 or Master Scheduler ABEND occurs)

- Tables/Work Areas: XSA, CIB, STAE control block

- Attributes: Reenterable, read-only, transient, self-relocatable

- Control Section: IEE0003D

IEE00110: SVC 110 - M/S Router

This routine passes control to the display processors, for DISPLAY and MONITOR commands that involve status displays, and the DUMP command processor, for main storage image dump.

- Entry: IEE00110 (this routine is the first load of SVC 110; SVC 110 is issued by IEEPALTR and IEEVDSP1)

- Exit: IEE10110, IEE20110, IEE30110, IEE40110, IEE50110, IEE60110

- Attributes: Reentrant, transient

- Tables/Work Areas: CSCB, M/S Resident Data Area, XSA

IEE0303D: SVC 34 - Chain Manipulator Routine

This module performs CSCB and CIB chain manipulation. When SVC 34 is entered for command processing, this routine exits to the Command Translator routine (module IEE5403D).

- Entry: IEE0303D

- Exit: To IEE5403D for command processing or return to caller

- Table/Work Areas: CVT, M/S resident data area, CSCB, XSA, CIB, LCE (TSO), TSCVT (TSO)

- Control Sections: IEE0303D

IEE0403D: SVC 34 - Router Routine

The purpose of this routine is to identify the command verb, and to pass control to the appropriate routine.

- Entry: IEE0403D

- Exit: Depending on command verb, via XCTL to another SVC 34 module

- Tables/Work Areas: M/S resident data area, XSA, CSCB, UCM, UCME

- Control Sections: IEE0403D

IEE0503D: SVC34 - Message Assembly Routine

The purpose of this module is to assemble and edit messages for the Command Scheduling routine, and to write the messages to the console or TSO terminal operator.

- Entry: IEE0503D

- Exit: Branch on register 14

- Tables/Work Areas: XSA

- Attributes: Reentrant, read-only, self-relocating

- Control Section: IEE0503D

IEE0603D: SVC 34 - SET Command Routine Part I

The purpose of this module is to process the SET command parameters CLOCK and DATE.

- Entry: IEE0603D

- Exit: To IEE8603D or IEE0503D

- Tables/Work Areas: XSA, CVT, M/S resident data area

- Attributes: Reentrant, read-only, self-relocating, transient

- Control Sections: IEE0603D

IEE0703D: SVC 34 STOP and MODIFY Scheduling Routine

This module schedules the execution of the MODIFY and nonperiodic STOP commands by finding and updating the appropriate CSCB, and issuing a POST macro instruction to the master scheduling task.

- Entry: IEE0703D

- Exit: To IEE0503D or return to caller

- Tables/Work Areas: M/S Resident Data Area, XSA, CVT, CSCB, CIB

- Control Sections: IEE0703D

IEE0803D: SVC34 - CSCB Creation Routine

The purpose of this routine is to schedule the execution of task-creating commands by adding a CSCB to the CSCB chain and issuing a POST macro instruction to the master scheduling task.

- Entry: IEE0803D

- Exit: Branch on register 14, XCTL to IEE3803D, IEE0503D, IEE2103D, IEE3803D, IEE7903D

- Tables/Work Areas: XSA, M/S resident data area, CVT and CSCB

- Attributes: Reentrant, partially disabled

- Control Sections: IEE0803D

IEE0903D: SET Command Handler

The purpose of this module is to process the date and time operands of the SET command.

- Entry: IEAQOT00

- Exit: SVC 3

- Tables/Work Areas: CVT, TSCVT (TSO)

- Attributes: Reentrant, supervisor state, disabled for system interrupts, transient

- Control Sections: IEAQOT00

IEE1A03D: SVC 34 - MCS Reply Processor Routine

The purpose of this routine is to process valid operator replies to WTOR macro instructions in an MCS environment.

- Entry: IEE1A03D

- Exit: To IEE1B03D

- Control Sections: IEE1A03D

- Tables/Work Areas: CVT, RQE, UCM, WQE, XSA

IEE1B03D: SVC 34 - Reply Message Routine

This routine assembles, edits, and broadcasts the accepted reply to a WTOR macro instruction for the MCS Reply Processor routine (module IEE1A03D) of the Command Scheduling routine, and writes error messages to the operator whose command is in error in either an MCS or a non-MCS environment.

- Entry: IEE1B03D, from IEE1A03D for MCS, or from IEE1203D if not MCS

- Exit: Return to the caller of SVC 34

- Control Sections: IEE1B03D

- Tables/Work Areas: CVT, RQE, UCM, WQE, XSA

IEE10110: SVC 110 - DISPLAY C,K Routine (Load 1)

This routine begins processing of the DISPLAY C,K command by building the title and label lines of the display and writing them to the operator's console by means of the WTO macro instruction.

- Entry: IEE10110 from IEE00110 (SVC 110 issued by IEEPALTR)

- Exit: IEE11110

- Attributes: Reentrant, privileged, transient

- Tables/Work Areas: (CSCB, XSA)

IEE1103D: SVC 34 - VARY and UNLOAD Syntax Scan Routine

The purpose of this routine is to examine the command and its operand for syntax errors.

- Entry: IEE1103D

- Exit: IEEMPCKR, IEE2303D, IEE3103D, IEE0503D

- Tables/Work Areas: XSA, CVT, master scheduler resident data area, UCM

- Attributes: Reentrant, self-relocating, read-only

- Control Sections: IEE1103D

IEE11110: SVC 110 - DISPLAY C,K Routine (Load 2)

This routine continues processing of the DISPLAY C,K command by building the middle portion of the display and writing the lines to the operator's console by means of the WTO macro instruction.

- Entry: IEE11110 from IEE10110

- Exits: IEE12110

- Attributes: Reentrant, privileged, transient

- Tables/Work Areas: CSCB, XSA

IEE1203D: SVC 34 -- Reply Processor Routine

This routine processes valid operator replies to WTOR macro instructions in a non-MCS environment.

- Entry: IEE1203D

- Exit: To IEE1B03D to issue error messages or to return to the caller of SVC 34

- Control Sections: IEE1203D

- Tables/Work Areas: CVT, RQE, UCM, WQE, XSA

- Attributes: Reenterable, Supervisor State, Disabled

IEE12110: SVC 110 - DISPLAY C,K Routine (Load 3)

This routine completes building of the DISPLAY C,K status display.

- Entry: IEE12110

- Exits: Return to caller (IEE00110)

- Attributes: Reentrant, privileged, transient

- Tables/Work Areas: CSCB, XSA

IEE1403D: SVC 34 - HALT and SWITCH Commands Processor Routine

This routine schedules the execution of the HALT and SWITCH commands. For the HALT command, this routine issues SVC 76 to pass control to the Statistics Update routine (see the I/O Supervisor PLM). For the SWITCH command, this routine issues SVC 83 to pass control to the SMF SVC routine (see part 6 of this publication).

- Entry: IEE1403D

- Exit: SVC 3

- Tables/Work Areas: XSA, M/S resident data area, CVT, CSCB

- Attributes: Reenterable

- Control Sections: IEE1403D

IEE1603D: See the MVT Supervisor PLM

IEE20110: SVC 110 - DISPLAY U Routine (Load 1)

This routine begins processing of the DISPLAY U command by checking the command syntax and locating the UCB for the first device to be included in the status display.

- Entry: IEE20110 from IEE00110 (SVC 110 issued by IEEPALTR)

- Exit: IEE21110, IEE22110, IEE23110

- Attributes: Reentrant, privileged, transient

- Tables/Work Areas: CSCB, Workarea obtained by GETMAIN, UCB

IEE2103D: SVC 34 - Message Assembly Routine

The purpose of this routine is to assemble and edit messages for the Command Scheduling routine, and to write the messages to the console or TSO terminal operator.

- Entry: IEE2103D

- Exit: Branch on register 14

- Tables/Work Areas: XSA

- Control Sections: IEE2103D

IEE21110: SVC 110 - DISPLAY U Routine (Load 2)

This routine continues processing of the DISPLAY U command by building display lines using information obtained from the UCB.

- Entry: IEE21110
- Exit: IEE23110, and IEE22110
- Attributes: Reentrant, privileged, transient
- Tables/Work Areas: Work Area obtained by GETMAIN, UCB, Device Name Table

IEE22110: SVC 110 - DISPLAY U Routine (Load 3)

This routine issues error and information pertaining to the DISPLAY U commands. It also builds line entries for data cell entries in the display and performs final cleanup (freeing the workarea) before returning to IEE00110.

- Entry: IEE22110 from IEE20110, IEE23110, or IEE21110
- Exit: IEE23110 or IEE22110, or return to IEE00110 via XCTL
- Attributes: Reentrant, privileged, type 4 SVC
- Tables/Work Areas: CSCB, UCB, Work Area from GETMAIN

IEE2303D: SVC 34 - SMF VARY Record Handler Routine

The purpose of this routine is to build and issue an SMF record for each device placed in online status.

- Entry: IEE2303D
- Exit: IEE3103D, IEE4203D, or IEE4403D
- Tables/Work Areas: CVT, SMCA, XSA
- Attributes: Reentrant, read-only, self-relocating
- Control Sections: IEE2303D

IEE23110: SVC 110 - DISPLAY U Routine (Load 4)

This routine continues processing of the DISPLAY U command by locating UCBs that satisfy the command and by writing lines of the display (created by loads 2 and 3) to the operator's console.

- Entry: IEE23110 from IEE20110, IEE21110 and IEE22110
- Exits: IEE21110 and IEE22110
- Attributes: Reentrant, privileged, transient
- Tables/Work Areas: UCB, CSCB, Work Area from GETMAIN

IEE2903D: SVC 34 - Display Requests Routine

The purpose of this routine is to display (to the requesting console or TSO terminal operator) the ID of all outstanding WTORs, the unit name of each device for outstanding MOUNT messages, and an indication as to whether any AVR mount messages are pending.

- Entry: IEE2903D, from IEE3503D
- Exit: Return to caller
- Tables/Work Areas: Message work area, XSA
- Attributes: Reentrant, Refreshable
- Control Sections: IEE2903D

IEE30110: SVC 110 - DISPLAY M Routine (Load 1)

This routine processes DISPLAY M commands that request CPU and channel status.

- Entry: IEE30110 from IEE00110 (SVC 110 issued by IEEPALTR)
- Exits: IEE31110, IEE32110
- Attributes: Reentrant, transient
- Tables/Work Areas: (CSCB, XSA; Work Area from GETMAIN; IEAQFX, MP65, low main storage

IEE3103D: SVC 34 - VARY and UNLOAD Processor Routine

The purpose of this routine is to process: the UNLOAD command, all VARY command operands in a system without the MCS option, and VARY ONLINE and OFFLINE operands for non-console devices in a system with the MCS option.

- Entry: IEE3103D, from IEE1103D, IEE2303D, IEE4203D, and IEE4603D
- Exit: IGC1703D, IEE4203D, IEE4303D, IEE4403D, IEE4703D, IEE0503D, return to caller
- Tables/Work Areas: XSA, UCM, CVT, M/S resident data area, and UCB

- Attributes: Reentrant, self-relocating, read-only

- Control Sections: IEE3103D

IEE31110: SVC 110 - DISPLAY M Routine (Load 2)

This routine builds the display of device and/or storage status requested by the DISPLAY M command.

- Entry: IEE31110

- Exit: IEE32110

- Attributes: Reentrant, transient

- Tables/Work Areas: CVT, IOSGEN Map; CSCB, IEAQFX, MP65, low main storage, work area from GETMAIN, XSA

IEE3203D: SVC 34 - VARY Keyword Router Routine

The purpose of this routine is to identify the first keyword in a VARY command, check its validity, and pass control to the appropriate command keyword processing routine.

- Entry: IEE3203D from IEE0403D

- Exit: To IEE1103D, IEE3303D, IEE4303D, IEE4703D, IEE2403D, IEE1703D

- Tables/Work Areas: XSA

- Attributes: Reentrant, read-only, self-relocating

- Control Sections: IEE3203D

IEE32110: SVC 110 - DISPLAY M Routine (Load 3)

This routine writes to the operator's console the lines of the display built by IEE30110 and IEE31110. It also issues error messages pertaining to the DISPLAY M status display.

- Entry: IEE32110 from IEE30110 and IEE31110

- Exit: IEE30110, IEE31110 and IEE00110

- Attributes: Reentrant, transient

- Tables/Work Areas: XSA, CSCB, work area from GETMAIN

IEE3303D: SVC 34 - MCS VARY Syntax Check Routine

The purpose of this routine is to scan syntax in an MCS environment.

- Entry: IEE3303D, from IEE3203D

- Exit: IEE2303D, IEE2203D, IEE4203D, or IEE4403D

- Tables/Work Areas: XSA, UCB, UCM, CVT

- Attributes: Reentrant, self-relocating, read-only

- Control Sections: IEE3303D

IEE3503D: SVC 34 - DISPLAY/MONITOR Commands Router Routine

The purpose of this routine is to examine and identify the operand of the DISPLAY and MONITOR commands and route the command to the appropriate task that will process the command.

- Entry: IEE3503D, from IEE0403D

- Exit: XCTL to IEE0503D, IEE1303D, IEE0803D, IEE2103D, IEE2903D, IEE7503D, IEE5803D, IEE8503D, or return to caller

- Tables/Work Areas: M/S resident data area, XSA, CVT, UCM, TSCVT (TSO), TJB (TSO)

- Attributes: Reentrant, transient, relocatable

- Control Sections: IEE3503D

IEE3603D: M65MP Vary Pre-processor Routine

This module prevents devices that have no paths marked available to the I/O Supervisor from becoming 'online'.

- Entry: IEE3603D, from IEE3203D

- Exit: IEE1103D, IEE3303D, Branch on Register 14

- Tables/Work Areas: IOSGEN, Message Work Area

- Attributes: Reentrant, Supervisor State, Disabled

- Control Section: IEE3603D

IEE3703D: SVC 34 - CANCEL Processor Routine

The purpose of this routine is to check the CANCEL command for syntax errors and scan the CSCB chain. If the command indicates cancellation of a current job that is processing and the routine finds the CSCB, the routine marks the CSCB cancellable and posts the master scheduling task. If no CSCB exists, the routine passes control via an XCTL macro instruction to the CSCB Creation routine.

- Entry: IEE0703D
- Exit: XCTL to IEE0803D or IEE0503D or IEE2103D or return to caller
- Tables/Work Areas: CVT, XSA, M/S resident data area, CSCB
- Control Sections: IEE3703D

IEE3803D: SVC 34 - START/MOUNT Hierarchy Parameter Processor Routine

The purpose of this module is to scan the START/MOUNT command operand field for the hierarchy parameter and set appropriate bits in the CSCB.

- Entry: IEE3803D from IEE0803D
- Exit: To IGC0503D and return to caller
- Tables/Work Areas: XSA, CSCB, CVT, M/S Resident Data Area
- Attributes: Reentrant, SVC transient, self-relocating, read only
- Control Sections: IEE3803D

IEE40110: SVC 110 - DISPLAY PFK Routine

This routine builds the display of PFK definitions, requested by the DISPLAY PFK command, and writes the display to the operator's console.

- Entry: IEE40110
- Exit: IEE00110 (for return to supervisor)
- Attributes: Reentrant, transient
- Tables/Work Areas: DCM

IEE4103D: SVC 34 - Hardcopy Message Issuing Routine

The purpose of this routine is to issue messages concerning the status of the hard copy log.

- Entry: IEE4103D, from IEE4703D
- Exit: Return to caller
- Tables/Work Areas: XSA, message area, UCB, CVT, XSA, and UCM
- Attributes: Reentrant
- Control Sections: IEE4103D

IEE4203D: SVC 34 - VARY Secondary Keyword Scan Routine

The purpose of this module is to perform authority and operand validity checking, and to pass control to the routine that will process the command.

- Entry: IEE4203D, from IEE3303D or IEE2303D
- Exit: To IEE3103D, IEE4603D, IEE4903D
- Tables/Work Areas: XSA, CVT, UCM, and UCB
- Attributes: Reentrant, self-relocating, read-only
- Control Sections: IEE4203D

IEE4303D: SVC 34 - VARY MSTCONS Processor Routine

The purpose of this routine is to process the VARY MSTCONS command.

- Entry: IEE4303D, from IEE3203D
- Exit: To IEE0503D, IEE2103D, IGCSL07B return to caller
- Tables/Work Areas: UCB, CVT, XSA and UCM
- Attributes: Reentrant, self-relocating, read-only
- Control Sections: IEE4303D

IEE4403D: SVC 34 - VARY CONSOLE/HARDCPY Keyword Scan Routine

The purpose of this routine is to check the validity of VARY CONSOLE/HARDCPY keywords, and to set appropriate bits in the XSA.

- Entry: IEE4403D, from IEE3303D or IEE2303D
- Exit: To IEE4203D, IEE0503D, IEE2103D
- Tables/Work Areas: XSA, UCM, CVT, and UCB
- Attributes: Reentrant, transient
- Control Sections: IEE4403D

IEE4503D: SVC 34 - STOP/STOPMN Command Handler Routine

The purpose of this routine is to process the commands STOP JOBNAMES/STATUS/SPACE/DSNAME and MODIFY.

- Entry: IEE4503D, from IEE0403D
- Exit: To IEE0703D, IEE0503D, IEE2103D, IEE5503D, IEE7503D and return to caller
- Tables/Work Areas: XSA, M/S Resident data area, CVT, UCM, UCME, TJB (TSO), TSCVT (TSO)
- Attributes: Reentrant, self-relocating, read-only
- Control Sections: IEE4503D

IEE4603D: SVC 34 - VARY ONLINE/OFFLINE for Console Devices Routine

The purpose of this routine is to process VARY ONLINE/OFFLINE for all MCS consoles.

- Entry: IEE4603D, from IEE4203D
- Exit: IEE3103D or return to caller
- Tables/Work Areas: XSA, CVT, UCB, UCM, and M/S resident data area
- Attributes: Reentrant, self-relocating, read-only
- Control Sections: IEE4603D

IEE4703D: SVC 34 - VARY HARDCOPY Processor Routine

The purpose of this routine is to process VARY HARDCPY commands.

- Entry: IEE4703D, from IEE3203D
- Exit: To IEE5703D or IEE7203D
- Tables/Work Areas: XSA, UCM, M/S resident data area, CVT, UCB
- Attributes: Reentrant, Transient
- Control Sections: IEE4703D

IEE4803D: SVC 34 - VARY CONSOLE Information Message Routine

The purpose of this routine is to construct a message that shows the current status of the varied console.

- Entry: IEE4803D, from IEE4903D
- Exit: IEE7203D, IEE7303D, return to caller
- Tables/Work Areas: XSA, message area, UCB, CVT, and UCM
- Attributes: Reentrant, transient
- Control Sections: IEE4803D

IEE4903D: SVC 34 - VARY CONSOLE Processor Routine

The purpose of this module is to process the VARY CONSOLE command.

- Entry: IEE4903D, from IEE4203D
- Exit: To IEE4803D, IEE4603D
- Tables/Work Areas: XSA, CVT, UCB, UCM
- Attributes: Reentrant, self-relocating, read-only
- Control Sections: IEE4903D

IEE50110: SVC 110 - DISPLAY Active Message Processor

This routine processes the work area initialized by IEEVDSP1 to produce the display of active masks.

- Entry: IEE50110 from IEE00110 (SVC 110 issued by IEEVDSP1)
- Exit: Return to caller (IEEVDSP1)
- Attributes: Reentrant, privileged
- Tables/Work Areas: CVT, UCM, XSA

IEE5103D: SVC 34 - STAE Exit Routine (Load 1)

The purpose of this routine is to scan the CSCB, CIB, GCB, and LOGON communication element (LCE) chains for errors. If a pointer is found to be in error, the routine truncates the chain by setting the pointer to zero. If possible, the routine provides a dump of main storage.

- Entry: IEE5103D from IEE0003D
- Exit: To IEE5203D
- Tables/Work Areas: CSCB, CIB, GCB, LCE, XSA, ABEND/STAE work area
- Attributes: Reenterable, self-relocating, transient
- Control Sections: IEE5103D

IEE5203D: SVC 34 - STAE Exit Routine (Load 2)

The purpose of this routine is to scan the reply queue element (RQE) chain and to truncate it if any block in the chain is not in the system queue area (SQA) or it is not on a doubleword boundary. If either IEEVDSP1 or IEEVWAIT terminates abnormally, the routine frees the main storage obtained by the routines. The routine also turns off the display active bit in the M/S resident data area if IEEVDSP1 terminates abnormally.

- Entry: IEE5203D from IEE5103D
- Exit: XCTL to IEE5303D
- Tables/Work Areas: RQE, M/S resident data area, XSA
- Attributes: Reenterable, self-relocating, transient
- Control Sections: IEE5203D

IEE5303D: SVC 34 - STAE Exit Routine (Load 3)

The purpose of this routine is to translate and print error codes, ABEND completion codes, and PSWs to the issuing console at ABEND time. The routine also passes control to the appropriate retry routine.

- Entry: IEE5303D from IEE5203D
- Exit: To the ABEND/STAE interface routine to schedule the appropriate retry routine in IEE5303D, IEEVWAIT, or IEEVDSP1.
- Tables/Work Areas: XSA, ABEND/STAE work area
- Attributes: Reenterable, self-relocating, transient
- Control Sections: IEE5303D

IEE5403D: SVC 34 - Command Translator Routine

This module translates lower case letters (except those within apostrophes) into upper case.

- Entry: IEE5403D
- Exit: To IEE0403D
- Tables/Work Areas: CVT, XSA, UCM
- Control Sections: IEE5403D

IEE5503D: SVC 34 MCS/TSO Periodic STOP Command Processor (STOPMN Command Handler -- Load 2)

The purpose of this routine is to process the periodic STOP command when issued from a TSO terminal or in an MCS environment.

- Entry: IEE5503D
- Exit: IEE6703D, branch on register 14
- Tables/Work Areas: XSA, IEEBASEA, UCM, UCME, TJB, TSCVT, CVT
- Control Sections: IEE5503D
- Attributes: Reentrant, Read-Only, Self-relocating

IEE5603D: SVC 34 - MSGRT and CONTROL (MR and K) Message Module (Load 1)

This routine issues error messages resulting from routing location (MSGRT or L = xxx) errors and errors involving the CONTROL command.

- Entry: IEE5603D
- Exit: IEE5903D, return to caller
- Tables/Work Areas: XSA, DCM, UCM
- Attributes: Reentrant, transient, type 4 SVC

IEE5703D: SVC 34 - VARY HARDCPY, OFF Routine

The purpose of this routine is to vary the hardcopy log out of the system and to route any error messages to the appropriate error message modules.

- Entry: IEE5703D from IEE4703D
- Exit: To IEE0503D, IEE2103D
- Tables/Work Areas: XSA, UCM, UCB, CVT, M/S Resident Data Area
- Attributes: Reentrant, transient
- Control Sections: IEE5703D

IEE5903D: SVC 34 - MSGRT and CONTROL Message Module (Load 2)

This routine issues error messages resulting from routing location (L=xxx), message routing defaults (MSGRT command), or CONTROL command errors. This module completes processing that was begun by IEE5603D.

- Entry: IEE5903D from IEE5603D

- Exit: Return to caller

- Attributes: Reentrant, transient, type 4 SVC

- Tables/Work Areas: XSA, DCM, UCM

IEE60110: Master Scheduler -- DUMP Command Processor Routine

This routine handles DUMP commands entered from the operator's console.

- Entry: IEE60110

- Exit: IEE7903D

- Attribute: Reentrant, Read-only

- Control Section: IEE60110

- Tables/Work Areas: XSA, CSCB

IEE6303D: SVC 34 - MSGRT Routine (Load 1)

This routine processes default routing specifications entered by means of the MSGRT command (with the exception of MSGRT REF, which is handled by IEE6403D).

- Entry: IEE6303D

- Exit: XCTL to IEE6403D or return to caller

- Tables/Work Areas: UCM, XSA, MRCT

- Attributes: Reentrant, transient, type 4 SVC

- Control Section: IEE6303D

IEE6403D: SVC 34 - MSGRT Command (Load 2)

This routine processes the MSGRT REF command and constructs the required display.

- Entry: IEE6403D from IEE6303D

- Exits: IEE5603D or return to caller

- Attributes: Reentrant, transient, type 4 SVC

- Tables/Work Areas: UCM, XSA, DCM, RCT

IEE5803D: Command Router Routine

This module routes control to CSCB creation routine IEE0803D.

- Entry: IKJ5803D

- Exit: To IEE0803D to create the CSCB

- Tables/Work Areas: XSA

- Control Sections: IKJ5803D

- Attributes: Reusable

IEE6503D: SVC 34 - SET Time-of-Day (TOD) Clock Routine - Part I

This routine performs TOD clock processing for System/370 models. It calculates the time at midnight, the difference between the current time and the time being set, and it passes these values to module IEE6603D.

- Entry: IEE6503D

- Exit: SVC 3 or XCTL to IEE6603D, IKJEAI00(TSO)

- Tables/Work Areas: CVT, XSA, M/S resident data area

- Attributes: Reenterable, SVC transient, disabled for system interruptions

- Control Sections: IEE6503D

IEE6603D: SVC 34 - TOD Clock Routine - Part II (TQE Update Routine)

This routine sets the TOD clock, updates the date in the CVT, updates TSO and system TQEs, and issues a message indicating that a SET command has been processed.

- Entry: IEE6603D

- Exit: SVC 3

- Tables/Work Areas: CVT, XSA, M/S resident data area

- Attributes: Reenterable, SVC transient, disabled for system interruptions

- Control Sections: IEE6603D

IEE6703D: SVC 34 - CONTROL Command Handler (Load 1)

This routine performs syntax check and processing for the following commands: STOPMN; CONTROL E; CONTROL D; CONTROL S; CONTROL V; and CONTROL N. It identifies and passes control to other modules for processing these commands: CONTROL A; CONTROL M; and CONTROL C. It also passes control to an error message module (IEE5603D) if an error is encountered.

- Entry: IEE6703D from IEE7503D

- Exits: IEE7703D, IEE7803D, IEE5603D, or return to caller.

- Attributes: Refreshable, privileged, type 4 SVC

- Tables/Work Areas: XSA, DCM

IEE6803D: SVC 34 - CONTROL Command Handler (Load 2)

This routine processes the CONTROL A command (except CONTROL A, REF, which is processed by IEE6903D)

- Entry: IEE6803D from IEE7803D

- Exit: IEE5603D or return to caller

- Attributes: Refreshable, privileged, type 4 SVC

- Tables/Work Areas: XSA, CVT, UCM, UCME, DCM SACB ID list (this table is constructed by this module)

IEE6903D: SVC 34 - CONTROL Command Handler (Load 5)

This routine handles CONTROL A, REF commands.

- Entry: IEE6903D from IEE7803D

- Exits: IEE5603D or return to caller

- Attributes: Refreshable, privileged, type 4 SVC

- Tables/Work Areas: DCM, UCM, XSA

IEE7103D: SVC 34 - MONITOR Command Router

This routine identifies the MONITOR command and routes control to the appropriate routine for further processing.

- Entry: IEE7103D from IEE3503D

- Exit: IEE0503D, IEE2103D, IEE7503D, or return to caller

- Attributes: Reentrant, transient

- Tables/Work Areas: XSA, CVT, UCM, M/S Resident Data Area

IEE7203D: SVC 34 - VARY HARDCOPY/UNIT Processor

This routine processes HARDCOPY or UNIT operands of the VARY command.

- Entry: IEE7203D from IEE4703D

- Exits: IEE4103D, IEE5703D

- Attributes: Reentrant, transient, privileged

- Tables/Work Areas: CVT, UCM, XSA, UCB, M/S Resident data area

IEE7303D: SVC 34 - MCS Console Information (Load 2)

This routine constructs messages displaying the console configurations resulting from VARY CONSOLE commands.

- Entry: IEE7303D from IEE4803D

- Exits: IEE4803D or return to caller

- Attributes: Reentrant, transient

- Tables/Work Areas: XSA, UCB, UCM, UCME, Buffer Core, DCM (including the SACBs)

IEE7503D: SVC 34 - Routing Location Routine (Load 1)

This routine processes location operands (L = xxx) and message routine defaults for CONTROL, STOPMN, DISPLAY and MONITOR commands:

- Entry: IEE7503D from IEE3503D, IEE7103D, IEE4503D, IEE0403D

- Exits: IEE6703D, IEE7603D, IEE5603D

- Tables/Work Areas: XSA, MRCT, UCM, Resident DCM (including SACBs), Master Scheduler Resident Data Area

- Attributes: Reentrant, refreshable, privileged, type 4 SVC

- Control Section: IEE7503D

IEE7603D: SVC 34 - Routing Location Module (Load 2)

This routine continues the processing of the routing location operands (L=xxx) of the DISPLAY, MONITOR or STOPMN commands. This processing starts with module IEE7503D.

- Entry: IEE7603D from IEE7503D

- Exits: IEE0803D, IEE5503D, or return to caller

- Attributes: Reentrant, refreshable, privileged, type 4 SVC

- Tables/Work Areas: XSA, UCM, Resident DCM (including the SACBs), M/S Resident Data Area

IEE7703D: SVC 34 - CONTROL Command Handler (Load 3)

This routine processes CONTROL M commands.

- Entry: IEE7703D from IEE6703D
- Exit: IEE5603D or return to caller
- Attributes: Refreshable, privileged, type 4 SVC
- Tables/Work Areas: UCM, WQE, XSA

IEE7803D: SVC 34 - CONTROL Command Handler (Load 4)

This routine processes CONTROL C commands.

- Entry: IEE7803D from IEE6703D
- Exit: IEE6803D, IEE6903D, IEE5603D or return to caller
- Attributes: Reentrant, transient, type 4 SVC
- Tables/Work Areas: XSA, UCM, WQE

IEE7903D: SVC 34 -- DUMP Message Module

This routine assembles and edits messages for the DUMP command processor. It also writes the messages to an operator's (console).

- Entry: IEE7903D from IEE60110 or IEE0803D
- Exit: Branch on (contents of) register 14
- Attribute: Reentrant
- Control Section: IEE7903D

IEE8503D: SVC 34 -- DISPLAY SQA Routine

This routine displays the low and high boundaries of the system queue area and the amount of free storage contained within the area.

- Entry: IEE8503D, from IEE3503D
- Exit: Return to caller
- Tables/Work Areas: Master Scheduler Resident Data Area, XSA, CVT
- Attributes: Reentrant
- Control Section: IEE8503D

IEE8603D: SVC 34 - SET Command Routine Part II

This module processes the SET command parameters: Q, PROC, and AUTO.

- Entry: IEE8603D
- Exit: To IEE0903D, IEE6503D, IEE0503D, or return to caller
- Tables/Work Areas: XSA, CVT, Master Scheduler resident data area
- Attributes: Reenterable, self-relocating, read-only, transient
- Control Section: IEE8603D

IEFACT: Interpreter - Dummy Accounting Routine

The purpose of this routine is to take the place of the user's accounting routine when none is specified.

- Entry: IEFACT
- Exit: Return to caller
- Attributes: Reentrant
- Control Sections: IEFACT

IEFACTFK: Termination - Dummy Accounting Routine

The purpose of this routine is to take the place of the User's Accounting routine when a User Accounting routine was specified at system generation, but none was supplied.

- Entry: IEFACTLK
- Exit: Return to caller
- Attributes: Refreshable
- Control Sections: IEFACTLK

IEFACTLK: Termination - User Accounting Routine Linkage Routine

The purpose of this routine is to provide linkage between the termination routine and the user's accounting routine, and to set up the required parameter list and read the first record of the account control table.

- Entry: IEFACTLK
- Exit: To User's Accounting routine, IEESMFWI, return to caller

- Tables/Work Areas: LCT, JCT, SCT, JACT, SACT, QMPA, JMR, TCT
- Control Sections: IEFACTLK

IEFACTRT: Termination - Dummy Accounting Routine

The purpose of this routine is to take the place of the User-Supplied Accounting routine.

- Entry: IEFACTRT
- Exit: Return to caller
- Control Sections: IEFACTRT

IEFATECB: Allocate/IEFVPOST Communications Block

The purpose of this module is to provide a communication area between the I/O device allocation and termination in the MVT environment.

- Attributes: Nonexecutable, member of resident nucleus
- Control Sections: IEFQMWR

IEFAVFAK: I/O Device Allocation - Linkage Module

This routine passes control to the AVR routine (IEFXV001) via an XCTL macro instruction.

- Entry IEFXV001
- Exit: XCTL to IEFXV001
- Control Section: IEFXV001

IEFCNVRT: Queue Management Convert Routine

The purpose of this routine is to convert the address of a record from the NN form to the MBBCCHHR form.

- Entry: CNVRT
- Exit: Return to caller
- Tables/Work Areas: CVT, QMRES
- Attributes: Reentrant
- Control Sections: IEFCNVRT

IEFCVFAK: I/O Device Allocation - Linkage to IEFMCVOL

The purpose of this routine is to pass control to one of three entries in the Mount Control Volume routine (IEFMCVOL).

- Entry: IEFCVOL1, IEFCVOL2, IEFCVOL3
- Exit: To entry point IEFCVOL1, IEFCVOL2, or IEFCVOL3 in IEFMCVOL via XCTL
- Control Sections: IEFCVOL1

IEFDSDRP: Data Set Descriptor Record Processing Routine

The purpose of this routine is to process the job queue information in the DSDR record to make a restarting job's queue entry reflect the environment at checkpoint.

- Entry Point: IEFDSDRP
- Exit: Return to caller
- Attributes: Reentrant
- Tables/Work Areas: JCT, SCT, SIOT, JFCB, TIOT, UCB, CVT, VOLT, TCB, QMPA, CSCB, DCBD, DCB, JFCBX, SCTX, LCT
- Control Sections: IEFVDSDRP

IEFDSLST: Initiator (Data Set Integrity) - Enqueue List Construction

The purpose of this routine is to use the tree table built by module IEFDSTBL to build an enqueue parameter list containing all data set names in the tree, and to free the main storage occupied by the tree.

- Entry: IEFDSLST, from IEFSD161
- Exit: Branch on register 14
- Tables/Work Areas: Data set name tree table
- Attributes: Reentrant
- Control Sections: IEFDSLST

IEFDSOAL: Direct System Output Determination Routine

The purpose of this routine is to modify SYSOUT SIOT and JFCB in a job step that uses direct system output facilities.

- Entry: IEFDSOAL from IEFVMLS1
- Exit: To IEFVMLS1, IEFQBVMS
- Tables/Work Areas: CVT, LCT, QMPA, DSOCB, JCT, SCT, SIOT JFCB, UCB, BASEA
- Attributes: Reentrant
- Control Sections: IEFDSOAL

IEFDSOCP: Direct System Output Control Routine

The purpose of this routine is to initialize the direct system output facilities by constructing the DSOCB.

- Entry: IEFDSOCP
- Exit: XCTL to IEFDSOLP or return to caller
- Tables/Work Areas: BASEA, TCB, DSOCB, TIOT, JFCB, UCB, CVT, ECB/IOB, QMPA
- Attributes: Reentrant
- Control Sections: IEFDSOCP

IEFDSOCR: Direct System Output Environment Check Routine

The purpose of this routine is to check the SIOTs for a restarting job step to determine if DSO is required. If so, the routine checks the available DSOCBs to determine if the appropriate DSOCBs are available. If additional DSOCBs are required, the routine informs the operator.

- Entry: IEFDSOCR
- Exit: IEFSD162
- Tables/Work Areas: VCT, LCT, JCT, DSOCB, SIOT, DNT, QMPA
- Attributes: Reentrant
- Control Sections: IEFDSOCR

IEFDSOFB: Direct System Output Free Block Routine

The purpose of this routine is to free DSOCBs allocated to a job that is to be suspended. The routine also posts the DSOCB ECB if a STOP or MODIFY command is pending.

- Entry: IEFDSOFB from IEFDSOCR, IEFSD168, or IEFSD518
- Exit: Branch on register 14
- Tables/Work Areas: LCT, DSOCB, CVT, BASEA, JCT
- Attributes: Reentrant
- Control Sections: IEFDSOFB

IEFDSOLP: Direct System Output Wait Routine

The purpose of this routine is to release storage occupied by DSO and to wait for a STOP or MODIFY comamnd to be issued.

- Entry: IEFDSOLP
- Exit: To IEFDSOSM
- Attributes: Reentrant
- Control Sections: IEFDSOLP

IEFDSOSM: Direct System Output STOP and MODIFY Processing Routine

The purpose of this routine is to process the STOP and MODIFY commands for direct system output facilities.

- Entry: IEFDSOSM
- Exit: XCTL to IEFDSOLP or SVC 3
- Tables/Work Areas: Reentrant
- Attributes: Reentrant
- Control Sections: IEFDSOSM

IEFDSOWR: Direct System Output Writer Routine

The purpose of this routine is to write job separators, if they are called for by the direct system output procedure, and system messages, if the message class is specified for direct system output. The routine writes the job separators and system messages to the respective direct system output devices at the appropriate times during job processing.

- Entry: IEFDSOWR from IEFSD162, IEFDS31Q
- Exti: To IEFSD162, IEFSD31Q, IEFQMRAW, IEFSD094 or the user Job Separator routine
- Tables/Work Areas: TCB, LCT, QMPA, DSOCB, JCT, SCT, UCB, CVT, TIOT, DCB, SMB, BASEA
- Attributes: Reentrant
- Control Sections: IEFDSOWR

IEFDSTBL: Initiator (Data Set Integrity) - Tree Table Build Routine

The purpose of this routine is to build a table of unique data set names and their attributes (shared or exclusive).

- Entry: IEFDSTBL, from IEFSD161
- Exit: Branch on register 14
- Attributes: Reentrant
- Control Sections: IEFDSTBL

IEFDSTRT: Initiator (Data Set Integrity) - Table Build Routine Translate Table

The purpose of this module is to contain the translate table used by the Tree Table Build routine.

- Attributes: Reusable
- Control Sections: IEFDSTRT

IEFIDMPM: Message Module

The purpose of this module is to contain the messages used by the Indicative Dump routine.

- Attributes: Non-executable
- Control Sections: IEFIDMPM

IEFIDUMP: Indicative Dump Routine

The purpose of this routine is to format indicative dump information and place it in SMBs for output at step termination.

- Entry: IEFIDUMP
- Exit: To IEFYNIMP, IEFYSVMS
- Attributes: Reusable
- Tables/Work Areas: IEFIDMPM
- Control Sections: IEFIDUMP

IEFIIC: MVT Initiator-Interface Control Routine

The purpose of this routine is to construct the interface for the Initiator subroutine. This routine is the first module to receive control when the initiator is started.

- Entry: IEFIIC
- Exit: XCTL to IEFSD160
- Attributes: Reentrant
- Tables/Work Areas: IEL, IEZCOM
- Control Sections: IEFIIC

IEFIRC: Interpreter - Reader Control Routine

The purpose of this routine is to construct the interface for the Interpreter subroutine.

- Entry: IEFIRC
- Exit: XCTL to IEFIRCB or return to caller on error
- Attributes: Reentrant
- Tables/Work Areas: NEL, TIOT, CIB, SMCA
- Control Sections: IEFIRC

IEFIRCB: Interpreter - Linkage Routine

The purpose of this routine is to pass the NEL (constructed by IEFIRC) to IEFVH1.

- Entry: IEFIRCB
- Exit: Return to caller of IEFIRC
- Tables/Work Areas: NEL
- Attributes: Reentrant
- Control Sections: IEFIRCB

IEFKGFAK: MVT Interpreter Dummy Module

The purpose of this module is to prevent the occurrence of an unresolved external reference to module IEFKG during system generation.

- Attributes: Non-executable
- Control Sections: IEFKG

IEFK1MSG: Message Module

The purpose of this module is to contain the messages issued by the volume attribute setting routine (module IEFPRES).

- Entry: IEFK1MSG
- Attributes: Non-executable
- Control Sections: IEFK1MSG

IEFLOCDQ: Queue Management Dequeue By Jobname Routine

The purpose of this routine is to search a queue for a job or list of jobs, and to return information, dequeue, or cancel the specified jobs when found.

- Entry: LOCDQ, LOCCAN, LOC
- Exit: Return to caller
- Tables/Work Areas: QCR, LTH
- Attributes: Reentrant
- Control Sections: IEFLOCDQ

IEFMCVOL: I/O Device Allocation - Mount Control Volume Routine

The purpose of this routine is to have a control volume mounted when a data set called for in a job step cannot be located on any currently mounted control volume.

- Entry: IEFCVOL1, IEFCVOL2, IEFCVOL3
- Exit: IEFVMMS1, IEFVMFAK, IEFVMLS6, IEFYNIMP (IEF41FAK)
- Tables/Work Areas: LCT, JCT, SCT, SIOT, JFCB, VOLT, QMPA, UCB
- Control Sections: IEFCVOL1, IEFCVOL2, IEFCVOL3

IEFORMAT: MVT Queue Management - Queue Formatting Routine

The purpose of this routine is to place the work queue data set in the format required by the MVT queue management routines.

- Entry: IEFORMAT, from IEFSD055
- Exit: Return to IEFSD055
- Tables/Work Areas: DCB, DEB
- Attributes: Reusable
- Control Sections: IEFORMAT

IEFPRES: Volume Attribute Setting Routine

The purpose of this routine is to set the volume attributes of all volumes listed in the PRESRES data set, and of all unlisted, but permanently resident, direct access volumes.

- Entry: IEFPRES
- Exit: To IEEVPRES, IEEVIPL, or IEFK1MSG
- Tables/Work Areas: UCB
- Control Sections: IEFPRES

IEFPRTXX: Direct System Output - Tape to Printer or Card Punch Routine

The purpose of this routine is to transfer system output processed by DSO from a tape to a printer or a card punch.

- Entry: SPRINTER
- Exit: Return to caller
- Control Sections: SPRINTER

IEFQAGST: MVT Queue Management - Assign/Start Routine

The purpose of this routine is to set up an ECB/IOB and to prepare the queue manager parameter area for the assign routine.

- Entry: IEFQAGST
- Exit: Return to linkor
- Tables/Work Areas: Q/M resident data area, QMPA, CVT
- Attributes: Reentrant
- Control Sections: IEFQAGST

IEFQASGQ: MVT Queue Management - Assign Routine

The purpose of this routine is to assign records to a queue entry and to assign logical tracks as required.

- Entry: IEFQASGN
- Exit: Return to linkor
- Tables/Work Areas: Q/M resident data area, QMPA, CVT
- Attributes: Reentrant
- Control Sections: IEFQASGN, IEFQASNM

IEFQBVMS: MVT Queue Management - Control Routine

The purpose of this routine is to inspect the function code in the queue manager parameter area and, on the basis of this code, to branch to the appropriate queue management routines.

- Entry: IEFQMSSS
- Exit: To IEFQAGST, IEFQMRAW, IEFQMNQQ, or IEFQASGQ, return to linkor

- Tables/Work Areas: QMPA
- Attributes: Reentrant
- Control Sections: IEFQMSSS

IEFQDELQ: MVT Queue Management - Delete Routine

The purpose of this routine is to make those logical tracks assigned to a queue entry available for assignment to other queue entries.

- Entry: IEFQDELE
- Exit: Return to linkor
- Tables/Work Areas: LTH, QMPA; QCR, Q/M resident data area, CVT
- Attributes: Reentrant
- Control Sections: IEFQDELE

IEFQINTZ: MVT Queue Management - Get Region Routine for Queue Initialization

The purpose of this routine is to obtain a region for the queue initialization task and, when the task has been completed, to free the region.

- Entry: IEFQINTZ, via attach from IEEVIPL
- Exit: To IEFSD055 via LINK, to IEEVIPL via RETURN
- Attributes: Reentrant
- Control Sections: IEFQINTZ

IEFQMDQQ: MVT Queue Management - Dequeue Routine

The purpose of this routine is to remove the highest priority entry from the input queue or a system output queue.

- Entry: IEFQMDQ2
- Exit: Return to linkor
- Tables/Work Areas: CVT, Q/M resident data area, QCR, LTH
- Attributes: Reentrant
- Control Sections: IEFQMDQ2

IEFQMDUM: MVT Queue Management Dummy Module

The purpose of this module is to prevent the occurrence of an unresolved external reference to module IEFQMSSS during system generation.

- Attributes: Non-executable
- Control Sections: IEFQMSSS

IEFQMLK1: MVT Queue Management - Branch Routine

The purpose of this routine is to branch to the appropriate Queue Management routine on the basis of an assign or read/write function code issued by an initiator.

- Entry: IEFQMSSS
- Exit: To IEFQASGQ or IEFQMRAW
- Tables/Work Areas: QMPA
- Attributes: Reentrant
- Control Sections: IEFQMSSS

IEFQMNQQ: MVT Queue Management - Enqueue Routine

The purpose of this routine is to place an entry in the input queue or an output queue at the requested priority.

- Entry: IEFQMNQ2
- Exit: Return to linkor
- Tables/Work Areas CVT, Q/M resident data area, QMPA, QCR, LTH
- Attributes: Reentrant
- Control Sections: IEFQMNQ2

IEFQMRAW: MVT Queue Management - Read/Write Routine

The purpose of this routine is to perform the conversion of a TTR into a MBBCCHHR and to read or write up to 15 records of the work queue data set.

- Entry: IEFQMRAW
- Exit: Return to linkor
- Tables/Work Areas: Q/M resident data area, QMPA, CVT, IOB/ECB
- Attributes: Reentrant
- Control Sections: IEFQMRAW

IEFQMUNQ: MVT Queue Management - Unchain Routine

The purpose of this routine is to remove a task from the queue management no-work chain.

- Entry: IEFQMUNC

- Exit: Return to linkor
- Tables/Work Areas: CVT, Q/M resident data area, QCR
- Attributes: Reentrant
- Control Sections: IEFQMUNC

IEFQRESD: MVT Queue Management - Resident Main Storage Reservation Module

The purpose of this routine is to reserve 140 bytes of resident main storage for the queue-management-opened DCB/DEB and the master queue control record at NIP time.

- Attributes: Non-executable
- Control Sections: IEFJOB

IEFRCLN1: Restart Reader Linkage Routine

The purpose of this routine is to provide linkage between the Restart Reader and the Interpreter Initialization routine.

- Entry: IEFRCLN1
- Exit: To IEFVH1, IEFVRRC (Entry point IEFVRRCA)
- Attributes: Reentrant
- Control Sections: IEFRCLN1

IEFRCLN2 Restart Reader Linkage Routine

The purpose of this routine is to provide linkage between the Restart Reader and the Interpreter Initialization routine.

- Entry: IEFRCLN2
- Exit: To IEFVH1, IEFVRRC (Entry point IEFVRRCB)
- Attributes: Reentrant
- Control Sections: IEFRCLN2

IEFRDWRT: Queue Management RJE Read/Write Routine

The purpose of this routine is to read or write a QCR or a queue entry.

- Entry: RD,WRT
- Exit: Return to caller
- Tables/Work Areas: CVT
- Attributes: Reentrant
- Control Sections: IEFRDWRT

IEFRPREP: Restart Preparation Routine

The purpose of this routine is to determine whether a job step that has been abnormally terminated can be restarted.

- Entry: IEFRPREP
- Exit: Return to caller
- Attributes: Reentrant; Character dependence type C
- Tables/Work Areas: LCT, JCT, SCT, PDQ, QMPA
- Control Sections: IEFRPREP

IEFRSTRT: Restart SVC Issuing Routine

The purpose of this routine is to issue the Restart SVC.

- Entry: IEFRSTRT, IEFSMR
- Exit: SVC 52 (RESTART), Return to caller
- Attributes: Reentrant
- Control Sections: IEFRSTRT

IEFSDPPT: MVT Initiator - Program Properties Table Module

This nonexecutable module consists of a table of the names of jobs that are allowed privileged execution. The first part of the table is a list of the names of jobs that will receive the requested region size; the second part of the table is a list of the names of the jobs that are noncancellable (except during allocation). The Region Size Determination routine (module IEFSD101) scans the list during its processing.

IEFSDTTE: System Output Writer Printer Routine

The purpose of this routine is to analyze and identify parameters requiring change as a result of change to the UCS/FCB environment when the output device is a 3211 printer.

- Entry: IEFSDTTE
- Exit: Error-IEFSD089 Normal-to calling routine
- Tables/Work Areas: WKSOR, PARLIST
- Control Sections: IEFSDTTE

IEFSDXXX System Output Writer - Spanned Data Sets Handler

This routine handles variable-length data sets by dynamically moving the input segments into the output segments.

- Entry: IEFSDXXX
- Exit: Return to caller
- Tables/Work Areas: Control area for spanning, DCB
- Attributes: Reentrant
- Control Section: IEFSDXXX

IEFSDXYZ: System Output Writer - Command Chaining Access Method

This routine simulates the processing performed by the QSAM Put Locate and Truncate routines by using command chaining to write one string of up to nine buffers with a single IOB and a single EXCP macro instruction.

- Entry: IEFSDXYZ
- Exit: Return to caller
- Tables/Work Areas: ECB, IOB, I/O table
- Attributes: Reentrant
- Control Section: IEFSDXYZ

IEFSD0XX: MVT Initiator - Dummy Job Step Module

The purpose of this routine is to abnormally terminate the job with condition code 213, when the joblib or fetch data set could not be opened.

- Entry: IEFSD0XX
- Exit: Return via ABEND
- Attributes: Problem Program Mode
- Control Sections: IEFSD0XX

IEFSD017: MVT Termination - System Output Interface Routine

The purpose of this routine is to provide an interface between the termination entry routine and system output processing.

- Entry: IEFSD017
- Exit: To IEFSD42Q
- Control Sections: IEFSD017

IEFSD055: MVT Queue Management - Queue Initialization Routine

The purpose of this routine is to construct a resident DEB/DCB, to pass control to the queue formatting routine or the first phase of system restart, to initialize the queue manager resident data area, and (if required) to pass control to the second phase of the system restart routine. If the jobqueue of the UCB indicates that the queue is on a RPS device, routine IEFSD055 provides information that is used by other routines prior to reading from or writing to the jobqueue.

- Entry: IEFSD055, from IEFQINTZ
- Exit: To IEFORMAT, IEFSD300, or IEFSD305
- Attributes: Reusable
- Control Sections: IEFSD055

IEFSD070: System Output Writer - Data Set Writer Attach Routine

The purpose of this routine is to attach the data set writer task, thus passing control to the standard data set writer or to the user-supplied data set writer routine.

- Entry: IEFSD070
- Exit: To IEFSD087 or user-supplied routine via ATTACH, or to IEFSD171 via XCTL
- Attributes: Reentrant
- Control Sections: IEFSD070

IEFSD078: System Output Writer - Linkor Routine

The purpose of this routine is to determine whether the record obtained from the output queue entry is a DSB or SMB, and to pass control, accordingly, to the DSB or SMB processor.

- Entry: IEFSD078
- Exit: To IEFSD085, IEFSD086, or IEFSD079
- Attributes: Reentrant
- Control Sections: IEFSD078

IEFSD079: System Output Writer - Link to Queue Manager Delete Routine

The purpose of this routine is to delete the current output queue entry

- Entry: IEFSD079

- Exit: To IEFQDELQ and IEFSD082

- Tables/Work Areas: QMPA

- Attributes: Reentrant

- Control Sections: IEFSD079

IEFSD080: System Output Writer - Initialization Routine

The purpose of this routine is to initialize the system output writer by obtaining main storage for a parameter list and the output DCB.

- Entry: IEFSD080

- Exit: To IEFSD081

- Tables/Work Areas: DCB, CSCB, TIOT, JFCB

- Attributes: Reentrant

- Control Sections: IEFSD080

IEFSD081: System Output Writer - Class Name Setup Routine

The purpose of this routine is to obtain main storage for, and initialize, a list of ECB pointers, ECBs, and queue management communication elements, depending on the system output classes specified for the writer. The routine also opens the output DCB.

- Entry: IEFSD081

- Exit: To IEFSD082

- Tables/Work Areas: CSCB, ECB

- Attributes: Reentrant

- Control Sections: IEFSD081

IEFSD082: System Output Writer - Main Logic Routine

The purpose of this routine is to obtain main storage for QMPAs and internal work areas, to dequeue output queue entries, to check for operator commands, and to pass control to the appropriate routine.

- Entry: IEFSD082

- Exit: IEFSD083, IEFSD084, IEFSD078

- Tables/Work Areas: CSCB, ECB

- Attributes: Reentrant

- Control Sections: IEFSD082

IEFSD083: System Output Writer - Command Processing Routine

The purpose of this routine is to process MODIFY and STOP commands that apply to the writer.

- Entry: IEFSD083

- Exit: To IEFSD081 or IEEVTCTL.

- Tables/Work Areas: CSCB, DCB, QMPA, ECB

- Attributes: Reentrant

- Control Sections: IEFSD083, IEFSD83M

IEFSD084: System Output Writer - Wait Routine

The purpose of this routine is to wait for an entry to be enqueued in an output queue corresponding to a class available to the writer.

- Entry: IEFSD084

- Exit: To IEFSD082

- Attributes: Reentrant

- Control Sections: IEFSD084

IEFSD085: System Output Writer - DSB Handler Routine

The purpose of this routine is to initialize for data set processing, and to inform the operator of the pause option in effect.

- Entry: IEFSD085, IEF085SD, or IEF850SD

- Exit: To IEFSD070

- Attributes: Reentrant

- Control Sections: IEFSD085, IEFSD85M

IEFSD086: System Output Writer - SMB Handler

The purpose of this routine is to initialize for message processing, and to extract each message from the current SMB.

- Entry: IEFSD086, IEF086SD

- Exit: To IEFSD088, IEFSD089, IEFQMNQQ, IEFQMRAW, IEFSD085, IEFSD078

- Tables/Work Areas: SMB, UCB, QMPA, TIOT, CSCB, TCB

- Attributes: Reentrant

- Control Sections: IEFSD086, IEFSD86M

IEFSD087: System Output Writer - Standard Writer Routine

The purpose of this routine is to get records from a data set.

- Entry: IEFSD087
- Exit: To IEFSD088, IEFSD089, IEFSD078
- Tables/Work Areas: DCB
- Attributes: Reentrant
- Control Sections: IEFSD087, IEFSD87M

IEFSD088: System Output Writer - Transition Routine

The purpose of this routine is to handle the transition between messages and data sets, and between data sets.

- Entry: IEFSD088
- Exit: To IEFSD089
- Tables/Work Areas: DCB
- Attributes: Reentrant
- Control Sections: IEFSD088

IEFSD089: System Output Writer - PUT Routine

The purpose of this routine is to format records as required and issue the PUT macro instructions to write them on the output unit.

- Entry: IEFSD089
- Exit: To IEFSD088
- Tables/Work Areas: DCB
- Attributes: Reentrant
- Control Sections: IEFSD089, IEFSD89M

IEFSD094: System Output Writer - Job Separator Routine

The purpose of this routine is to print or punch a job name and system output class designation on the writer's output device.

- Entry: IEFSD094
- Exit: To IEFSD088, IEFSD089, IEFSD095, IEFSD078
- Control Sections: IEFSD094

IEFSD095: System Output Writer - Print Line Routine

The purpose of this routine is to construct the block letters used to separate jobs processed by a system output writer when the output data set is to be printed.

- Entry: IEFSD095
- Exit: To IEFSD094
- Attributes: Reentrant
- Control Sections: IEFSD095

IEFSD096: Message Module

The purpose of this module is to contain message headers and texts for messages to the operator.

- Entry: IEFSD096
- Attributes: Nonexecutable
- Control Sections: IEFSD096

IEFSD097: MVT I/O Device Allocation - Wait for Space Decision Routine

The purpose of this routine is to make the decision whether to wait for direct access space, and to provide an interface with the I/O Device Allocation Space Request routine so that re-try and additional recovery passes may be made.

- Entry: IEFSD097
- Exit: Branch on register 14
- Tables/Work Areas: LCT, TIOT, UCB
- Attributes: Read-only, reenterable
- Control Sections: IEFSD097

IEFSD101: MVT Initiator - Replace Region Interface Routine

The purpose of this routine is to determine the correct region size for a starting task.

- Entry: IEFSD101
- Exit: To IEFSD102, IEFUJI, IEFUSI
- Tables/Work Areas: CVT, LCT, SCT, M/S resident data area, JMR, JCT
- Attributes: Reentrant
- Control Sections: IEFSD101

IEFSD102: MVT Initiator - Replace Region Routine

The purpose of this routine is to free the current region, enqueue the data sets used in the current task , and obtain a new region for the current task.

- Entry: IEFSD102
- Exit: To IEFSD162
- Tables/Work Areas: LCT, data set enqueue parameter list
- Attributes: Reentrant
- Control Sections: IEFSD102

IEFSD103: MVT Initiator - Pre-Attach Routine

The purpose of this routine is to place the ICT for system tasks in the job queue, and to initialize the ATTACH macro instruction parameter list.

- Entry: IEFSD063
- Exit: To IEFSD263, IEFSD514
- Tables/Work Areas: LCT, CSCB, JSCB, WTPCB, SCT
- Attributes: Reentrant
- Control Sections: IEFSD063

IEFSD104: MVT Initiator - Post-Attach Interface Routine

The purpose of this routine is to take a post-invocation exit if one is specified in the initiator exit list, to read the LCT from the job queue for system tasks.

- Entry: IEFSD104
- Exit: To IEFSD164, IEFSD514
- Tables/Work Areas: TIOT, LCT, JSCB, WTPCB
- Attributes: Reentrant
- Control Sections: IEFSD104

IEFSD105: MVT Initiator - Initiator Wait Routine

The purpose of this routine is to free the current initiator region when there is no work for the initiator, to wait for a job to be enqueued or a STOP INIT command to be issued, and to obtain a new region for the initiator. This routine must be loaded into the link pack area.

- Entry: IEFSD105
- Exit: To IEFSD161
- Tables/Work Areas: CVT, CSCB, LCT, BASEA
- Attributes: Reentrant
- Control Sections: IEFSD105

IEFSD110: MVT Queue Management (Track Stacking) - Stack Initialization Routine

The purpose of this routine is to obtain and initialize an area for use in keeping a stack of tracks from the work queue data set in main storage.

- Entry: IEFSD110
- Exit: Return to caller
- Tables/Work Areas: Track stack area, QMPA, Q/M Resident data area, CVT
- Attributes: Reentrant
- Control Sections: IEFSD110

IEFSD111: MVT Queue Management (Track Stacking) - Record Accessing Program

The purpose of this routine is to transfer work queue data set records between the track stack and the user's record area, and to transfer logical tracks between the work queue data set and the track stack as required.

- Entry: IEFSD111
- Exit: Return to caller
- Tables/Work Areas: Track stack area, Q/M Resident data area, CVT
- Attributes: Reentrant
- Control Sections: IEFSD111

IEFSD112: MVT Queue Management (Track Stacking) - Stack Purge Program

The purpose of this routine is to transfer any updated logical tracks remaining in the stack to the work queue data set, and to release the main storage occupied by the track stack.

- Entry: IEFSD112
- Exit: Return to caller
- Tables/Work Areas: Track stack area, Q/M resident data area, CVT

- Attributes: Reentrant

- Control Sections: IEFSD112

IEFSD160: MVT Initiator Initialization Routine

The purpose of this routine is to initialize the "life-of-task" functions of the initiator.

- Entry: IEFSD060

- Exit: To IEFSD161 or the routine specified in the initiator exit list

- Tables/Work Areas: CVT, LCT, Q/M resident data area, QMPA, ECB list, GCB, initiator exit list

- Control Sections: IEFSD060, IEFSD60M

IEFSD161: MVT Initiator Job Selection Routine

The purpose of this routine is to obtain the highest priority job from the input queue if the Initiator subroutine has received control to initiate a job step task; to read the JCT and first SCT into main storage; to set up the message class QMPA; and to build a data set enqueue parameter list. In addition, this routine determines whether a STOP or MODIFY command has been issued and takes appropriate action. If DSO is in the system, the routine selects eligible DSOCBs for the job.

- Entry: IEFSD061

- Exit: To IEFSD263, IEFSD101, IEFSD105, IEFSD160, IEFQMDQQ, IEFQMRAW, IEFDSOSL, or the routine specified in the initiator exit list.

- Tables/Work Areas: JCT, SCT, LCT, QMPA, CSCB, SCD, IOB, DSENQ TABLE, CVT, ECB list, GCB, initiator exit list, JSCB, WTPCB, DSOCB, BASEA

- Attributes: Reentrant

- Control Sections: IEFSD061, QMTMSD, IEFDSOAL

IEFSD162: MVT Initiator - I/O Device Allocation Interface Routine

The purpose of this routine is to complete the entrance interface for the I/O Device Allocation routine, to pass control to that routine, and to determine whether the task can be attached. If a step is being restarted from a checkpoint, control passes to the DSO Environment routine. If the return code is normal, initiation continues; if it is not normal, control passes to the Job Selection routine to select another job. The routine also builds a DSO message and passes control via a LINK macro instruction to the DSO Writer routine to write system messages and/or job separators. The routine also writes the TIOT in the job queue.

- Entry: IEFSD062

- Exit: To IEFSD103, IEFSD164, IEFSD514, IEFSD21Q, IEFQMRAW, IEFDSOWR, IEFDSOCR, IEFSD161, pre-invocation if specified.

- Tables/Work Areas: LCT, JCT, SCT, TIOT, ATTACH macro instruction parameter list, user parameter list, JSCB, WTPCB, QMPA, SMB, DSOCB, BASEA

- Attributes: Read-only, reenterable

- Control Sections: IEFSD062

IEFSD164: MVT Initiator - Task Delete Routine

The purpose of this routine is to prepare the parameters required by the Termination routine after a job step or system task has been executed or when the return code from the I/O Device Allocation routine indicates that a step cannot be executed.

- Entry: IEFSD064

- Exit: To IEFSD101, IEFSD166, IEFSD168

- Tables/Work Areas: QMPA, LCT, supervisor parameter list, JCT, SCT, ACT, JSCB, WTPCB

- Attributes: Read-only, reenterable

- Control Sections: IEFSD064

IEFSD166: MVT Initiator - Job Delete Routine

The purpose of this routine is to delete a completed job from the input queue, and to release the main storage occupied by job related tables.

- Entry: IEFSD066

- Exit: To IEFSD161

- Tables/Work Areas: LCT, QMPA, Q/M resident data area, CSCB, CVT, JCT, JSCB, WTPCB

- Attributes: Reentrant

- Control Sections: IEFSD066

IEFSD168: Initiator - Job Suspension Routine

The purpose of this routine is to reenqueue a terminated job so that the job can be reactivated.

- Entry: IEFSD068
- Exit: To IEFSD161, IEFVSDRA, IEFDSOFB
- Tables/Work Areas: QMPA, LCT, JCT, SCD, SCT, JSCB, WTPCB
- Attributes: Reentrable
- Control Sections: IEFSD068

IEFSD171: System Output Writer - Data Set Delete Routine

The purpose of this routine is to obtain records from an output queue entry, and to delete system output data sets.

- Entry: IEFSD071
- Exit: To IEFQMNQQ, IEFSD085, IEFSD086, IEFSD078, or IEFQMRAW
- Tables/Work Areas: DCB, SMB, UCB, CVT, QMPA, TIOT, CSCB, TCB
- Attributes: Reentrant
- Control Sections: IEFSD071, IEFSD71M

IEFSD180: MVT I/O Device Allocation - Dedication Determination Routine

The purpose of this routine is to determine whether the data set described by a DD statement being processed is a dedicated data set, and if it is, to copy previously stored information about the dedicated data set into the SIOT and JFCB created from the DD statement.

- Entry: IEFSD18
- Exit: To IEFVMLS1
- Tables/Work Areas: LCT, SCT, SIOT, JFCB, PDQ, TIOT, AWA, CVT
- Control Section: IEFSD180

IEFSD195: MVT I/O Device Allocation - Wait for Unallocation Routine

The purpose of this routine is to provide the I/O device allocation the ability to wait for unallocation to occur during the execution of another task, when allocation cannot be completed because of current allocations.

- Entry: IEFVAWAT
- Exit: Return to caller
- Tables/Work Areas: JCT, SCT, SIOT, LCT, ECB, CSCB
- Attributes: Read-only, reenterable
- Control Sections: IEFSD095

IEFSD21Q: MVT I/O Device Allocation - MVT Allocation Entry Routine

The purpose of this routine is to provide an interface for entry to the I/O Device Allocation routine operating in the MVT environment.

- Entry: IEFW21SD
- Exit: To IEFVKIMP, IEFVMLS1 or IEFWD000
- Tables/Work Areas: JCT, LCT, SCT, SMB, QMPA, CVT, TJB (TSO), TSCVT (TSO)
- Attributes: Read-only, reenterable
- Control Sections: IEFW21SD, IEFR102

IEFSD22Q: Termination Routine - Step Terminate Exit Routine

The purpose of this routine is to provide an interface between the Termination routine and the Task Delete or Alternate Step Delete routine when a step has been terminated.

- Entry: IEFW22SD
- Exit: Return to caller of Termination routine
- Tables/Work Areas: JCT, SCT, SMB, LCT, QMPA, ECB
- Attributes: Read-only, reenterable
- Control Sections: IEFW22SD

IEFSD263: MVT Initiator - Attach Routine

The purpose of this routine is to pass control to the task via the ATTACH macro instruction. If a job step is canceled or the time limit is reached, an ABEND dump is provided.

- Entry: IEFSD263
- Exit: IEFSD104
- Tables/Work Areas: ATTACH macro instruction parameter list, user parameter list, TCT, TCTIOT, LCT

- Attributes: Reentrant
- Control Sections: IEFSD263

IEFSD300: MVT System Restart - Initialization Routine

The purpose of this routine is to read all QCRs and logical track header records into main storage, to build tables A, B, and C, and to remove from Table A all the LTH entries corresponding to logical tracks in the free-track queue or in one of the other queues.

- Entry: IEFSD300
- Exit: To IEFSD301
- Tables/Work Areas: system restart work area, Table A, Table B, Table C
- Attributes: Reentrant
- Control Sections: IEFSD300

IEFSD301: MVT System Restart - Purge Queue Construction Routine

The purpose of this routine is to search Table A for the last LTH corresponding to each queue entry, determine the type of entry, and construct the purge queue.

- Entry: IEFSD301
- Exit: To IEFSD302
- Tables/Work Areas: System restart work area, Table A, Table C, purge queue
- Attributes: Reentrant
- Control Sections: IEFSD301

IEFSD302: MVT System Restart - Jobnames Table Routine

The purpose of this routine is to remove from Table A, all logical tracks assigned to dequeued input, RJE, ASB, or hold queue entries as well as the logical tracks assigned to the output queue entries corresponding to dequeued and enqueued input queue entries. In addition, it builds a table of job names for incomplete input and RJE queue entries, for dequeued input queue entries, and dequeued ASB queue entries.

- Entry: IEFSD302
- Exit: To IEFSD303
- Tables/Work Areas: System restart work area, Table A, Table C, and the interpreter/initiator jobnames table
- Attributes: Reentrant
- Control Sections: IEFSD302

IEFSD303: MVT System Restart - Delete Routine

The purpose of this routine is to create a queue entry of the remaining logical tracks and to delete that entry, thus assigning those tracks to the free-track queue.

- Entry: IEFSD303
- Exit: IEFSD055
- Tables/Work Areas: System restart work area, QMPA, Table A
- Attributes: Reentrant
- Control Sections: IEFSD303

IEFSD304: MVT System Restart - Scratch Data Sets Routine

The purpose of this routine is to set up to scratch temporary data sets generated for incomplete or dequeued input queue entries.

- Entry: IEFSD304
- Exit: To IEFSD305, IEFSD308
- Tables/Work Areas: CVT, UCB address look-up table
- Attributes: Reentrant
- Control Sections: IEFSD304

IEFSD305: MVT System Restart - Reenqueue Routine

The purpose of this routine is to dequeue the entries in the purge queue and reenqueue them in the appropriate input, hold, RJE, or output queue and to inform the operator of the names of jobs in the process of interpretation and initiation.

- Entry: IEFSD305
- Exit: To IEFSD304, IEFVSDRA, IEFVSDRD
- Tables/Work Areas: System restart work area, purge queue, JCT, SCT, JFCB, DSB, SCD, SIOT.
- Attributes: Reentrant
- Control Sections: IEFSD305

IEFSD308: MVT System Restart - Scratch Data Sets Routine

The purpose of this routine is to scratch the temporary data sets generated for incomplete and dequeued input queue entries.

- Entry: IEFSD308
- Exit: To IEFSD304
- Tables/Work Areas: DSCB, DCB, UCB, CVT, VTOC, DEB
- Attributes: Reentrant
- Control Sections: IEFSD308

IEFSD31Q: Termination Routine - Job Termination Exit Routine

The purpose of this routine is to provide an interface between the Termination routine and the Step Delete or Alternate Step Delete routine when the last step of a job has been terminated. If the message class is specified for DSO, control passes to the DSO Writer routine. This routine also releases DSOCBs assigned to the job, and it posts the DSOCB if a STOP or MODIFY is pending for it.

- Entry: IEFW31SD
- Exit: To IEFDSOWR or return to caller of Termination routine
- Tables/Work Areas: JCT, SCT, SMB, QMPA, ECB, CVT, M/S resident data area, DSOCB, TJB (TSO), TSCVT (TSO)
- Attributes: Read-only, reenterable
- Control Sections: IEFW31SD

IEFSD310: MVT System Restart TTR and NN to MBBCCHHR Conversion Routine

The purpose of this routine is to convert a relative record address (NN) or a relative track and record address (TTR) to an actual disk address (MBBCCHHR).

- Entry: IEFSD310
- Exit: Return to linkor
- Tables/Work Areas: CVT
- Attributes: Reentrant
- Control Sections: IEFSD310

IEFSD311: MVT Queue Management - Message Module

The purpose of this module is to contain the messages required by the Queue Initialization routine (module IEFSD055).

- Entry: IEFSD311, SD55MSG1, SD55MSG2, SD55MSG3
- Attributes: Nonexecutable
- Control Sections: IEFSD311

IEFSD312: MVT System Restart - Message Module

The purpose of this module is to contain the messages required by the system restart routines.

- Entry: IEFSD312, SD304MG1, SD304MG2, SD305MG1
- Attributes: Non-executable
- Control Sections: IEFSD312

IEFSD41Q: MVT I/O Device Allocation - Allocation Exit Routine

The purpose of this routine is to provide an interface for exit from the I/O Device Allocation routine operating in an MVT environment.

- Entry: IEFW41SD
- Exit: To IEFVMLS1, or return to caller.
- Tables/Work Areas: JCT, LCT, SCT, SMB, QMPA, ATCA, MVCA, MVCAX
- Attributes: Read-only, reenterable
- Control Sections: IEFW41SD

IEFSD42Q: Termination Routine - Termination Entry Routine

The purpose of this routine is to provide an interface for entry to the Termination routine operating in an MVT environment.

- Entry: IEFW42SD
- Exit: To IEFYNIMP
- Tables/Work Areas: JCT, SCT, SMB, LCT, TIOT
- Attributes: Read-only, reenterable
- Control Sections: IEFW42SD

IEFSD447: RJE Write LTH Routine

The purpose of this routine is to set an indicator in the QMPA, and to write out the LTH.

- Entry: IEFSD447
- Exit: Return to caller
- Tables/Work Areas: CVT,QMRES,QMPA
- Attributes: Reentrant
- Control Sections: IEFSD447

IEFSD514: System Restart - Table Breakup Routine

This routine reads and writes tables which may be required by the job scheduler. The routine breaks the tables into 176-byte records, writes the records on disk, and retrieves the records from disk to reconstruct the tables in main storage.

- Entry: IEFSD514
- Exit: IEFQAGST, IEFQMRAW, or return to caller
- Tables/Work Areas: QMPA, TBR Parameter List
- Attributes: Read-only, reenterable
- Control Sections: IEFSD514

IEFSD551: I/O Device Allocation - Linkage to IEFXJIMP

This routine provides an interface linkage to IEFXJIMP via an XCTL macro instruction.

- Entry: IEFV15XL
- Exit: XCTL to IEFXJIMP
- Tables/Work Areas: Same as caller
- Attributes: Read-only, reenterable
- Control Section: IEFV15XL

IEFSD552: I/O Device Allocation - Linkage to IEFXJIMP

This routine provides an interface linkage to IEFXJIMP via an XCTL macro instruction.

- Entry: IEFXJX5A
- Exit: XCTL to IEFXJIMP
- Tables/Work Areas: Same as caller
- Attributes: Read-only, reenterable
- Control Section: IEFXJX5A

IEFSMFAT: Initiator - TCTIOT Construction Routine

The purpose of this routine is to construct the TCTIOT, append it to the TCT, initialize the TCT storage map, and store the user routine address in the TCT.

- Entry: IEFSMFAT
- Exit: Return to caller
- Tables/Work Areas: PQE, SMCA, TCB, TCT, TCTIOT, TIOT
- Attributes: Reentrant
- Control Sections: IEFSMFAT

IEFSMFIE: Initiator - User Exit Initialization Routine

The purpose of this routine is to initialize the parameter lists for the Job Initiation and Step Initiation user exits and to construct SMF type 20 records.

- Entry: IEFSMFIE
- Exit: Return to caller
- Tables/Work Areas: JCT, JMR, LCT, SCT, TCT, CVT, SMCA, ACT
- Attributes: Reentrant
- Control Sections: IEFSMFIE

IEFSMFLK: Termination Routine - User Exit Initialization Routine

The purpose of this routine is to initialize the parameter lists for the Job Termination and Step Termination user exits.

- Entry: IEFACTLK
- Exit: Return to caller
- Tables/Work Areas: JCT, JMR, LCT, SCT, SMCA, TCB, TCT
- Attributes: Reentrant
- Control Sections: IEFACTLK

IEFSMFWI: Termination Routine - SMF Writer Interface Routine

The purpose of this routine is to construct the SMF job termination and step termination records, and sign on/off messages for SYSOUT.

- Entry: IEFSMWI
- Exit: Return to caller
- Tables/Work Areas: JCT, JMR, LCT, SCT
- Attributes: Reentrant
- Control Sections: IEFSMFWI

IEFUJI: Initiator - Dummy User Job Initiation Exit Routine

The purpose of this routine is to simulate the presence of a user-supplied job initiation exit routine.

- Entry: IEFUJI
- Exit: Return to caller
- Attributes: Reentrant
- Control Sections: IEFUJI

IEFUJV: Interpreter - Dummy User JCL Validation Exit Routine

The purpose of this routine is to simulate the presence of a user-supplied JCL validation routine.

- Entry: IEFUJV
- Exit: Return to caller
- Attributes: Reentrant
- Control Sections: IEFUJV

IEFUSI: Initiator - Dummy User Step Initiation Exit Routine

The purpose of this routine is to simulate the presence of a user-supplied step initiation exit routine.

- Entry: IEFUSI
- Exit: Return to caller
- Attributes: Reentrant
- Control Sections: IEFUSI

IEFUSO: Initiator - Dummy Output Limit Routine

The purpose of this routine is to simulate the presence of a user-supplied Output Limit routine.

- Entry: IEFUSO
- Exit: Return to caller
- Attributes: Reentrant
- Control Sections: IEFUSO

IEFUTL: Dummy User Time Limit Exit Routine

The purpose of this routine is to simulate the presence of a user-supplied time limit exit routine.

- Entry: IEFUTL
- Exit: Return to caller
- Attributes: Reentrant
- Control Sections: IEFUTL

IEFVDA: Interpreter-DD Statement Processor

The purpose of this routine is to construct and add entries to a JFCB and SIOT from the complete logical DD statement in the internal text buffer.

- Entry: IEFVDA
- Exit: To IEFVHF
- Tables/Work Areas: IWA, LWA, SIOT, JFCB, JCB, SCT
- Attributes: Read-only, reenterable
- Control Sections: IEFVDA

IEFVDBSD: Interpreter Data Set Name Table Construction Routine

The purpose of this routine is to create a data set name table.

- Entry: IEFVDBSD
- Exit: To IEFVDA
- Attributes: Reentrant
- Control Sections: IEFVDBSD

IEFVEA: Interpreter - EXEC Statement Processor

The purpose of this routine is to construct or update an SCT, and, if necessary, a joblib JFCB and SIOT from the complete logical EXEC statement in the internal text buffer.

- Entry: IEFVEA, from IEFVFA
- Exit: To IEFVHF
- Tables/Work Areas: IWA, EXEC work area, interpreter key table, JCT, SCT, SIOT, QMPA, procedure override table.
- Attributes: Read-only, reenterable
- Control Sections: IEFVEA

IEFVFA: Interpreter - Scan Routine

The purpose of this routine is to scan the card image of a JOB, EXEC, or DD statement, perform error checking of JCL syntax, build internal text, and, when a complete logical statement (including continuations and overrides) has been scanned, to pass control to the appropriate statement processor.

- Entry: IEFVFA
- Exit: To IEFVGM, IEFVHQ, IEFVHF, IEFVJA, IEFVDA, IEFVEA, IEFVFA
- Tables/Work Areas: IWA, Scan routine work area, interpreter key table, QMPA, internal text buffer, scan dictionary, JMR, PDT
- Attributes: Read-only, reenterable
- Control Sections: IEFVFA

IEFVFB: Interpreter - Symbolic Parameter Processing Routine

The purpose of this routine is to process symbolic parameters by creating symbolic parameter table buffer entries to assign values to symbolic parameters, and to extract those values and place them in the intermediate text buffer when a symbolic parameter is used.

- Entry: IEFVFB
- Exit: Return to IEFVFA
- Tables/Work Areas: IWA, LWA, SYMBUF, Intermediate Text Buffer, QMPA
- Attributes: Read-only, reenterable
- Control Sections: IEFVFB

IEFVGI: Interpreter - Dictionary Entry Routine

The purpose of this routine is to construct entries for the refer-back dictionary.

- Entry: IEFVGI
- Exit: Return to caller
- Tables/Work Areas: Refer-back dictionary, auxiliary work area, IWA, QMPA
- Control Sections: IEFVGI

IEFVGK: Interpreter - Get Parameter Routine

The purpose of this routine is to search the internal text buffer for the next parameter, perform basic error checking, and pass control to the appropriate Keyword routine.

- Entry: IEFVGK
- Exit: Return to caller
- Tables/Work Areas: Local work area, IWA, internal text buffer, KBT, PDT.
- Control Sections: IEFVGK

IEFVGM: Interpreter - Message Processing Routine

The purpose of this routine is to construct SMBs containing interpreter error messages and JCL statement images, to assign space for these SMBs in the message class output queue entry, and to write the SMBs into the entry.

- Entry: IEFVGM
- Exit: Return to caller
- Tables/Work Areas: QMPA, SMB, SCD, IWA, JCT, JMR
- Attributes: Reentrant, character dependence type C
- Control Sections: IEFVGM

IEFVGM1: Interpreter - Message Module

The purpose of this module is to contain interpreter messages 01-07.

- Attributes: Nonexecutable
- Control Section: IEFVGM1

IEFVGM2: Interpreter - Message Module

The purpose of this module is to contain interpreter messages 08-0F.

- Attributes: Nonexecutable
- Control Sections: IEFVGM2

IEFVGM3: Interpreter - Message Module

The purpose of this module is to contain interpreter messages 10-17.

- Attributes: Nonexecutable
- Control Sections: IEFVGM3

IEFVGM4: Interpreter - Message Module

The purpose of this module is to contain interpreter messages 18-1F.

- Attributes: Nonexecutable
- Control Sections: IEFVGM4

IEFVGM5: Interpreter - Message Module

The purpose of this module is to contain interpreter messages 20-27.

- Attributes: Nonexecutable
- Control Sections: IEFVGM5

IEFVGM6: Interpreter - Message Module

The purpose of this module is to contain interpreter messages 28-2F.

- Attributes: Nonexecutable
- Control Sections: IEFVGM6

IEFVGM7: Interpreter - Message Module

The purpose of this module is to contain interpreter messages 30-37.

- Attributes: Nonexecutable
- Control Sections: IBFVGM7

IEFVGM8: Interpreter - Message Module

The purpose of this module is to contain interpreter messages 50-57.

- Attributes: Nonexecutable
- Control Sections: IEFVGM8

IEFVGM9: Interpreter - Message Module

The purpose of this module is to contain interpreter messages 58-5F.

- Attributes: Nonexecutable
- Control Sections: IEFVGM9

IEFVGM10: Interpreter - Message Module

The purpose of this module is to contain interpreter messages 60-67.

- Attributes: Nonexecutable
- Control Sections: IEFVGM10

IEFVGM11: Interpreter - Message Module

The purpose of this module is to contain interpreter messages 68-6F.

- Attributes: Nonexecutable
- Control Sections: IEFVGM11

IEFVGM12: Interpreter - Message Module

The purpose of this module is to contain interpreter messages 70-77.

- Attributes: Nonexecutable
- Control Sections: IEFVGM12

IEFVGM13: Interpreter - Message Module

The purpose of this module is to contain interpreter messages 78-7F.

- Attributes: Nonexecutable
- Control Sections: IEFVGM13

IEFVGM14: Interpreter - Message Module

The purpose of this module is to contain interpreter messages 88-8F.

- Attributes: Nonexecutable
- Control Sections: IEFVGM14

IEFVGM15: Interpreter - Message Module

The purpose of this module is to contain interpreter messages 90-97.

- Attributes: Nonexecutable
- Control Sections: IEFVGM15

IEFVGM16: Interpreter - Message Module

The purpose of this module is to contain interpreter messages A0-A7.

- Attributes: Nonexecutable
- Control Sections: IEFVGM16

IEFVGM17: Interpreter - Message Module

The purpose of this module is to contain interpreter messages 56-5D.

- Attributes: Nonexecutable
- Control Sections: IEFVGM17

IEFVGM18: Interpreter - Message Module

The purpose of this module is to contain interpreter messages 80-87.

- Attributes: Nonexecutable
- Control Sections: IEFVGM18

IEFVGM19: Interpreter Message Module

The purpose of this module is to contain interpreter messages 3E-45.

- Attributes: Nonexecutable
- Control Sections: IEFVGM19

IEFVGM70: Interpreter Message Module

The purpose of this module is to contain interpreter messages 38-3F.

- Attributes: Nonexecutable
- Control Sections: IEFVGM70

IEFVGM71: Interpreter - Message Module

The purpose of this module is to contain interpreter messages 40-47.

- Attributes: Nonexecutable
- Control Sections: IEFVGM71

IEFVGM78: Interpreter - Message Module

The purpose of this module is to contain interpreter messages 08-0D.

- Attributes: Nonexecutable
- Control Sections: IEFVGM78

IEFVGS: Interpreter - Dictionary Search Routine

The purpose of this routine is to search the refer-back dictionary for the address of a previously-defined SCT, SIOT, or JFCB.

- Entry: IEFVGS
- Exit: Return to caller
- Tables/Work Areas: Auxiliary work area, IWA, QMPA, refer-back dictionary
- Control Sections: IEFVGS

IEFVGT: Interpreter - Test and Store Routine

The purpose of this routine is to perform operations on a parameter as indicated in the appropriate parameter descriptor table entry.

- Entry: IEFVGT
- Exit: Return to Keyword routine
- Tables/Work Areas: Internal text buffer, PDT, local work area, IWA
- Control Sections: IEFVGT

IEFVHA: Interpreter (Control Routine) - Get Routine

The purpose of this routine is to read statements from the input stream and the procedure library.

- Entry: IEFVHA
- Exit: IEFVHC, IEFVHB, IEFVHAA, IEFVHR, IEFVGM
- Tables/Work Areas: IWA, JCT, DCB.
- Attributes: Read-only, reenterable
- Control Sections: IEFVHA

IEFVHAA: Interpreter (Control Routine) - End-of-File Routine

The purpose of this routine is to determine the conditions under which an end-of-file condition has occurred, and to set switches and pass control accordingly.

- Entry: IEFVHAA
- Exit: IEFVHC or IEFVHN
- Tables/Work Areas: IWA, JCT
- Attributes: Read-only, reenterable
- Control Sections: IEFVHAA

IEFVHB: Interpreter (Control Routine) - DD* Statement Generator Routine

The purpose of this routine is to generate a "SYSIN DD*" statement for data in the input stream, when no such statement was included.

- Entry: IEFVHB
- Exit: To IEFVHC, IEFVHA, IEFVGM
- Tables/Work Areas: IWA, JCT
- Attributes: Read-only, reenterable
- Control Sections: IEFVHB

IEFVHC: Interpreter (Control Routine) - Continuation Statement Routine

The purpose of this routine is to determine whether the current statement should be a continuation, and, if so, to determine whether it is a valid continuation statement.

- Entry: IEFVHC
- Exit: To IEFVHEB, IEFVHCB, IEFVGM
- Tables/Work Areas: IWA, JCT, DCB
- Attributes: Read-only, reenterable
- Control Sections: IEFVHC

IEFVHCB: Interpreter (Control Routine) - Verb Identification Routine

The purpose of this routine is to identify the verb on a control statement.

- Entry: IEFVHCB
- Exit: To IEFVHE, IEFVHM, IEFVHA, IEFVGM, IEFVHL, IEFVINA
- Tables/Work Areas: IWA, JCT
- Attributes: Read-only, reenterable
- Control Sections: IEFVHCB

IEFVHE: Interpreter (Control Routine) - Router

The purpose of this routine is to determine the conditions under which it was entered, and to pass control to the appropriate routine.

- Entry: IEFVHE
- Exit: To IEFVHEB, IEFVHH, IEFVHEC
- Tables/Work Areas: IWA
- Attributes: Read-only, reenterable
- Control Sections: IEFVHE

IEFVHEB: Interpreter (Control Routine) - Pre-Scan Preparation Routine

The purpose of this routine is to determine whether a message is required or additional work queue space is required before a statement is scanned. If so, it causes the message to be written or the work queue space to be assigned.

- Entry: IEFVHEB
- Exit: To IEFVHQ, IEFVGM, IEFVHG, IEFVFA, IEFUJV
- Tables/Work Areas: IWA, JCT, SCT, QMPA, JMR
- Attributes: Read-only, reenterable
- Control Sections: IEFVHEB

IEFVHEC: Interpreter (Control Routine) - Job Validity Check Routine

The purpose of this routine is to determine whether an SCT has been built for the current job; if not, the routine constructs an SCT.

- Entry: IEFVHEC
- Exit: To IEFVGM, IEFVHH
- Tables/Work Areas: IWA, JCT, SCT
- Attributes: Read-only, reenterable
- Control Sections: IEFVHEC

IEFVHF: Interpreter (Control Routine) - Post-Scan Routine

The purpose of this routine is to determine the conditions under which it was entered, and to pass control accordingly.

- Entry: IEFVHF
- Exit: To IEFVHG, IEFVHEB, IEFVHCB, IEFVHA
- Tables/Work Areas: IWA, CWA
- Attributes: Read-only, reenterable
- Control Sections: IEFVHF

IEFVHG: Interpreter (Control Routine) - Spool Routine

The purpose of this routine is to place an SIOT and a JCT into the input queue entry, and to write system input data sets to a direct access device.

- Entry: IEFVHG
- Exit: To IEFVHB, IEFVHR, IEFVGM, IEFVHQ, IEFVHAA, IEFVHA, IEFVHC
- Tables/Work Areas: IWA, JCT, SIOT, VOLT, CWA, SCT
- Attributes: Read-only, reenterable
- Control Sections: IEFVHG

IEFVHH: Interpreter (Control Routine) - Job and Step Enqueue Routine

The purpose of this routine is to place the SCT, DSNT, VOLT, and JCT in the job's queue entry, and to determine whether the interpreter is to enqueue jobs.

- Entry: IEFVHH
- Exit: To IEFKG, IEFVHN, IEFVHQ, or IEFVHHB
- Tables/Work Areas: IWA, JCT, NEL, QMPA, SCT
- Attributes: Read-only, reenterable
- Control Sections: IEFVHH

IEFVHHB: Interpreter (Control Routine) - Job and Step Enqueue Housekeeping Routine

The purpose of this routine is to initialize for merging a cataloged procedure.

- Entry: IEFVHHB
- Exit: To IEFVHA, IEFVHEB
- Tables/Work Areas: IWA
- Attributes: Read-only, reenterable
- Control Sections: IEFVHHB

IEFVHL: Interpreter (Control Routine) - Null Statement Routine

The purpose of this routine is to determine the conditions under which the null statement was encountered, and to pass control to the proper routine.

- Entry: IEFVHL
- Exit: To IEFVHCB, IEFHEC, IEFVHE, IEFVHA
- Tables/Work Areas: IWA, JCT
- Attributes: Read-only, reenterable
- Control Sections: IEFVHL

IEFVHM: Interpreter (Control Routine) - Command Statement Routine

The purpose of this routine is to test for valid command verbs, and, if the verb is valid, to issue SVC 34 to schedule execution of the command.

- Entry: IEFVHM
- Exit: To IEFVHA, IEFVGM
- Tables/Work Areas: IWA, JCT
- Attributes: Read-only, reenterable
- Control Sections: IEFVHM

IEFVHN: MVT Interpreter - Termination Routine

The purpose of this routine is to close the input stream and procedure library data sets, free main storage used by the interpreter, and build the interpreter exit list.

- Entry: IEFVHN
- Exit: Return to linkor
- Tables/Work Areas: IWA, JCT, QMPA, JMR
- Attributes: Read-only
- Control Sections: IEFVHN

IEFVHQ: Interpreter - Queue Management Interface Routine

The purpose of this routine is to be a common interface between the Queue Management routines and the interpreter.

- Entry: IEFVHQ
- Exit: Return to caller
- Tables/Work Areas: IWA, JCT, QMPA
- Attributes: Read-only, reenterable
- Control Sections: IEFVHQ

IEFVHR: Interpreter (Control Routine) - Operator Message Routine

The purpose of this routine is to write a message to the operator when an I/O error has occurred.

- Entry: IEFVHR
- Exit: Return to caller
- Tables/Work Areas: IWA, JCT, CWA, UCB
- Attributes: Read-only, reenterable
- Control Sections: IEFVHR

IEFVH1: Interpreter - Initialization Routine

The purpose of this routine is to initialize the interpreter; it obtains main storage for and initializes the IWA, local work areas; and the DSENQ table, it also processes the reader PARM field.

- Entry: IEFVH1
- Exit: To IEFVH2
- Tables/Work Areas: UCB, IWA, DCB, local work area, JMR
- Attributes: Nonreusable
- Control Sections: IEFVH1

IEFVH2: Interpreter - Initialization Open Routine

The purpose of this routine is to open the input stream data set and the procedure library data set, and to obtain main storage for a buffer for procedure library records.

- Entry: IEFVH2
- Exit: To IEFVHA
- Tables/Work Areas: IWA, UCB, TIOT
- Control Sections: IEFVH2
- Attributes: Nonreusable

IEFVINA: In-Stream Procedure Routing Routine

The purpose of this routine is to process an in-stream procedure by invoking other in-stream procedure modules as subroutines to perform the processing.

- Entry: IEFVINA
- Exit: To IEFVHA, IEFVHCB
- Tables/Work Areas: IWA, JCT, QMPA, in-stream procedure work area, in-stream procedure parameter list, IWA
- Attributes: Reenterable
- Control Sections: IEFVINA

IEFVINB: In-stream Procedure Directory Search Routine

The purpose of this routine is to search the in-stream procedure directory for the name of procedure that is to be processed.

- Entry: IEFVINB
- Exit: Return to caller
- Tables/Work Areas: In-stream procedure work area, in-stream procedure parameter list, IWA, QMPA
- Attributes: Reenterable
- Control Sections: IEFVINB

IEFVINC: In-stream Procedure Directory Build Routine

The purpose of this routine is to build a directory entry for each in-stream procedure.

- Entry: IEFVINC
- Exit: Return to caller
- Tables/Work Areas: IWA, QMPA, in-stream procedure work area, in-stream procedure parameter list
- Attributes: Reenterable
- Control Sections: IEFVINC

IEFVIND: In-stream Procedure Decompression Interface Routine

The purpose of this routine is to build a parameter list for the in-stream procedure, read a record from the job queue, and invoke the Record Decompression routine (module IEZDCODE) to decompress a record.

- Entry: IEFVIND
- Exit: Return to caller
- Tables/Work Areas: IWA, QMPA, in-stream procedure parameter list, in-stream procedure work area, DCB
- Attributes: Reenterable
- Control Sections: IEFVIND

IEFVINE: In-stream Procedure Syntax Check Routine

The purpose of this routine is to check the PROC and PEND statements for syntax errors.

- Entry: IEFVINE
- Exit: Return to caller
- Attributes: Reenterable
- Control Sections: IEFVINE

IEFVJA: Interpreter - Job Statement Processor

The purpose of this routine is to initialize a JCT and job ACT from the complete logical job statement in the internal text buffer.

- Entry: IEFVJA
- Exit: To IEFVHF
- Tables/Work Areas: IWA, job work area, interpreter key table, JCT, ACT, QMPA, PDT
- Attributes: Read-only, reenterable
- Control Sections: IEFVJA

IEFVJIMP: Termination - JOB Statement Condition Code Processor

The purpose of this routine is to test the condition codes specified in the JOB statement to determine whether the remaining steps in the job are to be run.

- Entry: IEFVJ
- Exit: To IEFVKIMP or IEFZAJB3
- Tables/Work Areas: LCT, JCT, SCT
- Control Sections: IEFVJ

IEFVJMSG: Termination - JOB Statement Condition Code Processor Messages

The purpose of this module is to contain the messages issued to the programmer by the JOB statement condition code processor.

- Entry: IEFVJMSG
- Attributes: Non-executable
- Control Sections: IEFVJMSG

IEFVKG: Interpreter (Control Routine) - Job and Step Enqueue Routing Routine

The purpose of this routine is to determine whether there are additional jobs to process.

- Entry: IEFVKG
- Exit: To IEFVHHB, IEFVHN
- Tables/Work Areas: IWA
- Attributes: Read-only, Reentrant
- Control Sections: IEFVKG

IEFVKIMP: I/O Device Allocation - EXEC Statement Conditional Execution Routine

The purpose of this routine is to determine, from the specifications in the EXEC statement COND field, whether the step should be bypassed or run.

- Entry: IEFVK
- Exit: IEFVS, IEFLB
- Tables/Work Areas: JCT, LCT, SCT
- Control Sections: IEFVK

IEFVKMSG: I/O Device Allocation - EXEC Statement Condition Code Processor Messages

The purpose of this module is to contain the messages issued to the programmer by the EXEC statement condition code processor.

- Entry: IEFVKMJ1
- Attributes: Non-executable
- Control Sections: IEFVKMSG

IEFVMA: Automatic SYSIN Batching - Initialization

The purpose of this routine is to acquire storage for and clear the ASBWA, process the START communications parameter area (CPA) and initialize data set processing.

- Entry: IEFVMA
- Exit: IEFVMB, IEFVMD, IEFVME
- Tables/Work Areas: ASBWA, JFCB, DCB, UCB, TIOT, CVT, CPA
- Attributes: Read-only
- Control Section: IEFVMA

IEFVMB: Automatic SYSIN Batching - Input Stream Processor

The purpose of this routine is to read an input stream, place SYSIN data sets on direct access volumes, and write JCL statements in the ASB queue.

- Entry: IEFVMB
- Exit: To IEFVMC, IEFVMD, IEFQAGST, IEFQMNQQ
- Tables/Work Areas: ASBWA, JFCB, DCB, UCB, TIOT, QMPA
- Attributes: Read-only
- Control Section: IEFVMB

IEFVMC: Automatic SYSIN Batching - Command Processor

The purpose of this routine is to determine whether unrecognizable input stream statements are command verbs, and if so, to process them according to the command disposition in the RDRA procedure.

- Entry: IEFVMC
- Exit: IEFVMB
- Tables/Work Areas: ASBWA
- Attributes: Read-only
- Control Section: IEFVMC

IEFVMD: Automatic SYSIN Batching - Termination

The purpose of this routine is to handle both normal and error termination conditions.

- Entry: IEFVMD
- Exit: To IEFQDELQ, return to System Task Control routine
- Tables/Work Areas: ASBWA, JFCB, DCB, UCB, TIOT, QMPA
- Attributes: Read-only
- Control Section: IEFVMD

IEFVME: Automatic SYSIN Batching - Interpreter Region Regulator

The purpose of this routine is to acquire a main storage region for execution of the Interpreter program, initiate interpretation, and free the main storage region when interpretation is completed.

- Entry: IEFVME
- Exit: Return to caller
- Attributes: Read-only
- Control Section: IEFVME

IEFVMF: Automatic SYSIN Batching - Interpreter Controller

The purpose of this routine is to construct the Interpreter entrance list (NEL) and to contain special access methods for reading JCL statements from the ASB queue and for finding cataloged procedures.

- Entry: IEFVMF
- Exit: To IEFQMDQQ, IEFQDELQ, IEFVME
- Tables/Work Areas: NEL, SAMWA
- Attributes: Read-only
- Control Section: IEFVMF

IEFVMG: Automatic SYSIN Batching - ASB Queue Reader

The purpose of this routine is to read and expand compressed JCL statements in the ASB queue.

- Entry: IEFVMG
- Exit: To IEFQMDQQ, IEFQMRAW, IEFQDELQ, return to caller
- Tables/Work Areas: SAMWA, DCB
- Attributes: Read-only
- Control Section: IEFVMG

IEFVMH: Automatic SYSIN Batching - Find

The purpose of this routine is to return to the caller the same information returned after execution of a FIND macro instruction from parameters in the SAMWA.

- Entry: IEFVMH
- Exit: Return to caller
- Tables/Work Areas: SAMWA, DCB
- Attributes: Read-only
- Control Section: IEFVMH

IEFVMFAK: I/O Device Allocation - Linkage to IEFVMLS1

The purpose of this routine is to pass control to control section IEFVMCVL of the JFCB Housekeeping routine.

- Entry: IEFVMCVL
- Exit: To IEFVMCVL via XCTL
- Control Sections: IEFVMCVL

IEFVMLS1: I/O Device Allocation - JFCB Housekeeping Control Routine and Allocate Processing Routines

The purpose of the Control routine is to obtain the required SIOTs, determine the processing required for each, and to pass control to the appropriate routine. The Allocate Processing routine performs the processing required in certain refer-back situations, when the data set is cataloged or passed, and when unit name is specified.

- Entry: IEFVM, IEFVMCVL, IEFVMQMI, VM7000, VM7030D, VM7055, VM7055AA, VM7060, VM7070, VM7090, VM7130, VM7370, VM7700, VM7742, VM7750, VM7850, VM7900, VM7950
- Exit: To IEFVM2LS, IEFVM3LS, IEFVM4LS, IEFVM5LS, IEFVM6LS, IEFXCSSS, IEFDSOAL
- Tables/Work Areas: LCT, JCT, PDQ, SIOT, JFCB, QMPA
- Control Sections: IEFVM1, IEFVMQMI, IEFVMPDQ, IEFVMVTE

IEFVMLS6: I/O Device Allocation - JFCB Housekeeping Error Message Processing Routine

The purpose of this routine is to prepare error messages for the JFCB Housekeeping routines.

- Entry: IEFVMSGR
- Exit: Return to caller
- Tables/Work Areas: JCT, LCT
- Control Sections: IEFVM6

IEFVMLS7: I/O Device Allocation - JFCB Housekeeping Error Messages

The purpose of this routine is to contain the messages issued by the JFCB Housekeeping routines.

- Entry: IEFVM7
- Attributes: Non-executable
- Control Sections: IEFVM7

IEFVMMS1: MVT Linkage to JFCB Housekeeping

The purpose of this routine is to provide a linkage to the JFCB Housekeeping routines for the graceful step flush function.

- Entry: IEFVM1
- Exit: To IEFVM1
- Attributes: Read-only, reenterable
- Control Sections: IEFVM1

IEFVM2LS: I/O Device Allocation - JFCB Housekeeping Fetch DCB Processing Routine

The purpose of this routine is to update the SIOT, SCT JFCB and VOLT with information required for the allocation of devices for the fetch DCB.

- Entry: VM7100
- Exit: To IEFVMLS1
- Tables/Work Areas: LCT, SCT, SIOT, JFCB, VOLT
- Control Sections: IEFVM2

IEFVM3LS: I/O Device Allocation - JFCB Housekeeping GDG Single Processing Routine

The purpose of this routine is to obtain the fully qualified name of a member of a GDG, and to complete the required information in the JFCB, VOLT, and SIOT for that member.

- Entry: VM7150
- Exit: To IEFVMLS1
- Tables/Work Areas: LCT, SIOT, GDG Bias Count table, JFCB
- Control Sections: IEFVM3

IEFVM4LS: I/O Device Allocation - JFCB Housekeeping GDG All Processing Routine

The purpose of this routine is to build an SIOT, JFCB, and VOLT, and PDQ entries for each member of the GDG.

- Entry: VM7200
- Exit: To IEFVMLS1
- Tables/Work Areas: LCT, SCT, VOLT, PDQ, SIOT, JFCB
- Control Sections: IEFVM4

IEFVM5LS: I/O Device Allocation - JFCB Housekeeping Patterning DSCB Routine

The purpose of this routine is to establish DCB control information within a JFCB.

- Entry: VM7300
- Exit: To IEFVMLS1
- Tables/Work Areas: LCT, SCT, SIOT, DSCB, JFCB
- Control Sections: IEFVM5

IEFVM76: I/O Device Allocation - JFCB Housekeeping Unique Volume ID Routine

The purpose of this routine is to create unique volume serials for unlabeled tape data sets, when the disposition is "PASS."

- Entry: VM7600
- Exit: Return to caller
- Tables/Work Areas: SIOT, JFCB, JFCBX
- Control Sections: IEFVM76

IEFVPOST: MVT I/O Device Allocation - Unsolicited Device Interrupt Handler

The purpose of this routine is to handle the posting of unsolicited device interruptions for I/O device allocation operating in an MVT environment.

- Entry: IEFDPOST
- Exit: To IEAOPT01 or return to caller
- Tables/Work Areas: ATCA, MVCA, MVCAX
- Attributes: Disabled, resident
- Control sections: IEFDPOST

IEFVRRC: Reinterpretation Control Routine

The purpose of this routine is to pass control among the routines that modify the queue entry of a restart step so that they appear as they were prior to the initiation of the step.

- Entry: IEFVRRC, IEFVRRCA, IEFVRRCB
- Exit: IEFRCLN1, IEFRCLN2, and return to caller
- Attributes: Read-only, Reentrant
- Tables/Work Areas: NEL, JCT, SCT, SIOT, JFCB, JFCBX, VOLT, SMB, DSENQ, SCD, DSB, QMPA
- Control Sections: IEFVRRC

IEFVRR1: Dequeue Interface Routine

The purpose of this routine is to interface with queue management to cause a specific job to be dequeued and the JCT for that job to be read into main storage.

- Entry: IEFVRR1
- Exit: Return to Caller
- Attributes: Read-only, Reentrant
- Tables/Work Areas: QMPA, JCT
- Control Sections: IEFVRR1

IEFVRR2: Table Merge Routine

The purpose of this routine is to merge the reinterpreted queue entry tables of a restart step with the original queue entry tables for that step.

- Entry: IEFVRR2, IEFV2AE
- Exit: Return to caller
- Attributes: Reentrant
- Tables/Work Areas: QMPA, JCT, ACT, SMB, SCT, SIOT, JFCB, DSENQ, VOLT, JFCBX, NEL
- Control Sections: IEFVRR2

IEFVRR3: Reinterpretation Delete/Enqueue Routine

The purpose of this routine is to delete the reinterpreted input and output queue entries of a restart step, to construct the internal JCL necessary for processing a checkpoint restart, and to reenqueue the job's queue entry.

- Entry: IEFVRR3, IEFV3AE
- Exit: Return to caller
- Attributes: Reentrant
- Tables/Work Areas: QMPA, JCT, SCT, SIOT, JFCB
- Control Sections: IEFVRR3

IEFVSCAN: Scan Routine

This routine scans the options buffer that is passed as input to the Job Scheduling Subroutine. It checks the buffer contents for correct syntax and gives control to (exits to) a specified subroutine for each valid buffer entry.

- Entry: IEFVSCAN, EIFVSCN1
- Exit: Return to caller
- Tables/Work Areas: None
- Attributes: Reentrant, Refreshable, Recursive
- Control Sections: IEFVSCAN

IEFVSDRA: Restart Activation Routine

The purpose of this routine is to issue a START Restart Reader command for one or more jobnames.

- Entry: IEFVSDRA
- Exit: Return to Caller
- Attributes: Reentrant
- Tables/Work Areas: CSCB, CVT, TCB
- Control Sections: IEFVSDRA

IEFVSDRD: Restart Determination Routine

The purpose of this routine is to initiate the process of automatic restart.

- Entry: IEFVSDRD
- Exit: To IEFSD305, IEFSD514, IEFSD42Q
- Attributes: Reentrant; Character dependence type C
- Tables/Work Areas: JCT, SCT, QMPA, CVT, TIOT, LCT
- Control Sections: IEFVSDRD

IEFVSD12: MVT Interpreter - SYSIN Processor Routine

The purpose of this routine is to set up a JFCB and allocate space on a direct access device for a system input data set.

- Entry: IEFSD012
- Exit: To IEFVHQ or return to caller
- Attributes: Reentrant
- Tables/Work Areas: IWA, DD work area, TIOT, UCB, JFCB
- Control Sections: IEFSD012

IEFVSD13: MVT Interpreter - SCD Construction Routine

The purpose of this routine is to construct an SCD entry for each system output class defined for a job, and to assign space for all DSBs that will be required.

- Entry: IEFSD090
- Exit: Return to caller
- Tables/Work Areas: IWA, QMPA, DD work area, SCD, SCT, SIOT, JCT, JFCB
- Control Sections: IEFSD090

IEFVSMBR: SMB Reader Routine

The purpose of this routine is to read the SMBs associated with a restarting job and to convert JCL statements to their original format.

- Entry: IGC0005B
- Exit: To IGC0105B, return to caller
- Attributes: Reentrant
- Tables/Work Areas: QMPA, DCB, JCT, SMB, RRCWKAR, SCT
- Control Sections: IGC0005B

IEFWA000: I/O Device Allocation - Demand Allocation Routine

The purpose of this routine is to establish data set device requirements, and to allocate in response to specific unit requests.

- Entry: IEFWA000, IEFUCBL
- Exit: To IEFWD000, IEFX3000, IEFX5000
- Tables/Work Areas: UCB Address List, DMT, UCB, LCT, SCT, SIOT, VOLT, AWT
- Control Sections: IEFWA7, IEFWA002, IEFUCBL

IEFWCFAK: I/O Device Allocation - Linkage Module

The purpose of this routine is to pass control to the TIOT Construction routine.

- Entry: IEFWC000

- Exit: To IEFWCIMP

- Control Sections: IEFWC000

IEFWCIMP: I/O Device Allocation - TIOT Construction Routine

The purpose of this routine is to calculate the main storage required for the TIOT, to build the TIOT, and to process requests for direct access space.

- Entry: IEFWC000

- Exit: To IEFXJIMP, IEFWD000

- Tables/Work Areas: JCT, SCT, LCT, SIOT, VOLT, AWT, TIOT

- Control Sections: IEFWC000, IEFWC002

IEFWDFAK: I/O Device Allocation: Linkage Module

The purpose of this module is to pass control to the External Action routine.

- Entry: IEFWD000

- Exit: To IEFWD000

- Control Sections: IEFWD000

IEFWD000: I/O Device Allocation - External Action Routine

The purpose of this routine is to verify that the correct direct access volumes are mounted in response to requests for volumes that require space allocation for new data sets. It also sets up an MVCA for the mounting of volumes containing old data sets.

- Entry: IEFWD000

- Exit: To IEFXT00D, IEFSD41Q, IEFXKIMP

- Tables/Work Areas: SCT, LCT, TIOT, UCB, ATCA, MVCA, MVCAX, CSCB, JFCB, TJB (TSO), TSCVT (TSO)

- Control Sections: IEFWD000, IEFWDMSG, IEFWD002

IEFWD001: I/O Device Allocation - External Action Messages

The purpose of this module is to contain a directory and the messages used in the External Action routine.

- Entry: IEFWD001

- Attributes: Nonexecutable

- Control Sections: IEFWD001

IEFWEXTA: I/O Device Allocation - Extended External Action Routine

The purpose of this routine is to verify that the correct direct access volumes are mounted in response to requests for volumes that contain old direct access data sets.

- Entry: IEFWEXTA

- Exit: To IEFW41SD, IEFXKIMP

- Tables/Work Areas: ATCA, MVCAX, CSCB, LCT, MVCA, SCT, UCB

- Control Sections: IEFWEXTA

IEFWSMSG: Termination - Message Module for Warmstart

The purpose of this module is to contain messages to be used by Module IEFWSYP3 during Warmstart.

- Entry: IEFWSMSG

- Attributes: Non-executable

- Control Sections: IEFWSMSG

IEFWSTRT: I/O Device Allocation - Message Module

The purpose of this module is to contain the messages issued to the operator when a job is started, and the messages issued to the operator when a job is terminated due to ABEND, condition codes, or JCL errors found any steps have been run.

- Entry: IEFWSTRT

- Attributes: Non-executable

- Control Sections: IEFWSTRT

IEFWSWIN: I/O Device Allocation - Linkage Module

The purpose of this module is to pass control to the Decision Allocation routine, or to the Automatic Volume Recognition routine.

- Entry: IEFWSWIT

- Exit: To IEFX5000 or IEFXV001

- Control Sections: IEFWSWIT

IEFWSYP3: Termination - SIOT Reader for Warmstart

The purpose of this module is either to match TIOT entries with SIOTs or, in the case of dynamically allocated data sets, to build TIOT entries for use by the termination routines.

- Entry: IEFWSYP3
- Exit: Return to caller
- Tables/Work Areas: QMPA, JCT, LCT, SIOT, UCB, TIOT, SIOTTTR, TIOTEXT
- Control Sections: IEFWSYP3

IEFWTERM: Termination - Message Module

The purpose of this module is to contain the messages issued to the operator when a job is terminated normally, or when it is terminated because of a JCL error found in the interpreter or initiator.

- Entry: IEFWTERM
- Attributes: Non-executable
- Control Sections: IEFWTERM

IEFWTP00: Write-to-programmer - Initialization Routine

The purpose of this routine is to initialize registers and also to obtain and initialize a work area before passing control to the WTP Message Processing routine (module IEFWTP01).

- Entry: IGC0203E from IGC0003E
- Exit: To IGC0003E, IGC0303E, or return to caller
- Tables/Work Areas: WTPCB, JSCB, UCM, CVT, IEFQMRES, IEFQMNGR, WPL
- Attributes: Reenterable
- Control Sections: IGC0203E

IEFWTP01: Write-to-programmer - Message Processing Routine

The purpose of this routine is to process write-to-programmer (WTP) messages. This routine uses the Transient Queue Management routine (SVC 90) to read, write, and assign SYS1.SYSJOBQE records for write-to-programmer messages.

- Entry: IGC0303E from IGC0203E and IGC0403E
- Exit: IGC0403E, IGC0003E, and return to caller
- Tables/Work Areas: WTPCB, JSCB, UCM, CVT, IEFQMRES, IEFQMNGR, WPL
- Attributes: Reenterable
- Control Section: IGC0303E

IEFWTP02: Write-to-programmer - Error Processing Routine

The purpose of this routine is to handle WTP processing when I/O errors occur in the job queue that is processing WTP messages, or when no record is available for a WTP message.

- Entry: IGC0403E from IGC0303E
- Exit: To IGC0003E, IGC0303E, or return to caller
- Tables/Work Areas: WTPCB, JSCB, UCM, CVT, IEFQMRES, IEFQMNGR, WPL
- Attributes: Reenterable
- Control Sections: IGC0403E

IEFXAMSG: I/O Device Allocation - Message Module

The purpose of this module is to contain the messages issued by the Allocation Control routine.

- Entry: IEFXAMSG
- Attributes: Non-executable
- Control Sections: IEFXAMSG

IEFXCSSS: I/O Device Allocation - Allocation Control Routine

The purpose of this routine is to calculate table space requirements and obtain the main storage for the tables used or built during allocation.

- Entry: IEFXA
- Exit: To IEFXJFAK, IEFWA000, IEFWCFAK
- Tables/Work Areas: JCT, SCT, LCT, SIOT, VOLT, AWT, UCB
- Control Sections: IEFXA, IEFXAB00

IEFXH000: I/O Device Allocation - Separation Strikeout Routine

The purpose of this routine is to strike from AWT entries, the bits corresponding to devices that would violate separation or affinity requests.

- Entry: IEFXH000
- Exit: Return to caller
- Tables/Work Areas: LCT, AWT, AVT, UCB
- Control Sections: IEFXH000

IEFXJFAK: I/O Device Allocation - Linkage Module

The purpose of this module is to pass control to the Allocation Recovery routine.

- Entry: IEFXJ000
- Exit: To IEFXJIMP
- Control Sections: IEFXJ000

IEFXJIMP: I/O Device Allocation - Allocation Recovery Routine

The purpose of this routine is to inform the operator of the allocation recovery options available, and to pass control to the proper routine to comply with his request.

- Entry: IEFXJ000, IEFV15XL, IEFXJX5A
- Exit: To IEFXCSSS, IEFSD095, IEFW41SD
- Attributes: Reusable
- Tables/Work Areas: LCT, AWT, JCT, CVT, UCB, SCT, SIOT
- Control Sections: IEFXJ000

IEFXJMSG: I/O Device Allocation - Allocation Recovery Messages

The purpose of this module is to contain the messages used by the Allocation Recovery routine.

- Entry: MSRCV, MSSYS, MSOFF
- Attributes: Non-executable
- Control Sections: IEFXJMSG

IEFXKIMP: I/O Device Allocation - Non Recovery Error Routine

The purpose of this routine is to cancel the step when a lack of available devices has been discovered after the TIOT is constructed.

- Entry: IEFXK000
- Exit: To IEFW41SD
- Tables/Work Areas: LCT, SCT, UCB, TIOT
- Control Sections: IEFXK000

IEFXKMSG: I/O Device Allocation - Non Recovery Error Routine Messages

The purpose of this routine is to contain the messages used by the Nonrecovery Error routine.

- Entry: IEFXKMSG
- Attributes: Non-executable
- Control Sections: IEFXKMSG

IEFXQM00: Transient Queue Management Initialization and Read/Write Routine

The purpose of this routine is to initialize tables and read or write job queue records.

- Entry: IGC00090
- Exit: To IGC01090 or return to caller
- Tables/Work Areas: Q/M resident data area, QMPA, CVT, ECB/IOB
- Attributes: Reenterable
- Control Sections: IGC00090

IEFXQM01: Transient Queue Management Track Assignment Routine

The purpose of this routine is to assign logical tracks as required.

- Entry: IGC01090
- Exit: To IGC02090 or return to caller
- Tables/Work Areas: QM resident data area, QMPA, CVT, ECB/IOB
- Attributes: Reenterable
- Control Sections: IGC01090

IEFXQM02: Transient Queue Management Record Assignment Routine

The purpose of this routine is to assign records to a queue entry.

- Entry: IGC02090
- Exit: Return to caller
- Tables/Work Areas: QM resident data area, QMPA, CVT, ECB/IOB
- Attributes: Reenterable
- Control Section: IGC02090

IEFXTDMY: Queue Management Error Return Routine

The purpose of this routine is to set up the correct error return code when the Queue Management Assign routine has been unable to assign a record for Initator routine use.

- Entry: IEFXTDMY
- Exit: to IEFXKIMP
- Attributes: Reentrant, reusable
- Tables/Work Areas: LCT
- Control Sections: IEFXTDMY

IEFXTMSG: I/O Device Allocation - Message Module

The purpose of this module is to contain the text of I/O Device allocation messages.

- Entry: MESGDDEV, MESGHEAD, MESGDSDV, RESMSG, RESUNIT, WTORREP, JOBCAN, PRIAMSG
- Attributes: Non-executable
- Control Sections: IEFXTMSG

IEFXT00D: I/O Device Allocation - Space Request Routine

The purpose of this routine is to obtain space on direct access devices for requesting data sets.

- Entry: IEFXT000, XTTBC0, WDEXIT
- Exit: To IEFWD000, IEFXT002, IEFXT003
- Tables/Work Areas: LCT, TIOT, UCB, JCT, SIOT, JFCB, PDQ
- Control Sections: IEFXT000

IEFXT002: I/O Device Allocation - TIOT Compression Routine

The purpose of this routine is to reduce the TIOT to its final size, and to provide an interface with the VARY command

- Entry: IEFXT002, XTTRDJ, XTTEB3, XTTEA1, XTTEA0
- Exit: To IEFWEXTA, IEFXKIMP, IEFXT003
- Tables/Work Areas: LCT, TIOT, UCB, JCT, SIOT, JFCB, PDQ
- Control Sections: IEFXT002

IEFXT003: I/O Device Allocation - DADSM Error Recovery Routine

The purpose of this routine is to determine the action to be taken when a request for direct access space is not honored.

- Entry: IEFXT003, XUUH06, XUUB00
- Exit: To IEFXT00D, IEFXT002
- Tables/Work Areas: LCT, TIOT, UCB, JCT, SIOT, JFCB
- Control Sections: IEFXT003

IEFXVMSG: I/O Device Allocation - AVR Routine Messages

The purpose of this routine is to contain the messages used by the AVR routine.

- Entry: IEFXVMSG
- Attributes: Non-executable
- Control Sections: IEFXVMSG

IEFXVNSL: I/O Device Allocation - AVR Nonstandard Label Processing Routine

The purpose of this routine is to take the place of the user's AVR Nonstandard Label Processing routine when none is supplied.

- Entry: IEFXVNSL
- Exit: Return to caller
- Attributes: Reentrant
- Control Sections: IEFXVNSL

IEFXV001: I/O Device Allocation - AVR Allocation Routine

The purpose of this routine is to perform allocation for specifically requested volumes.

- Entry: IEFXV001
- Exit: To IEFWCIMP, IEFWD000, IEFXJIMP, IEFXV002, IEFXV003, IEFX300A, IEFX5000
- Tables/Work Areas: ACB, AVT, AWT, JFCB, LCT, TIOT, UCB, VOLT
- Attributes: Reusable
- Control Sections: IEFXV001

IEFXV002: I/O Device Allocation - AVR Second Module

This module contains four subroutines that perform label processing and verification, affinity chain verification, serial list building, and issuing of AVR mount requests.

- Entry: IEFXV002, PATTEST, SERBUILD, PUTLIST
- Exit: To IEFXVNSL, Return to caller
- Tables/Work Areas: ACB, AVT, AWT, CVT, DEB, IOB, LCT, LUT, UCB, VOLT
- Attributes: Reusable
- Control Sections: IEFXV002

IEFXV003: I/O Device Allocation - AVR Unmounted Tape Allocation Routine

This routine determines the allocation solution for unmounted tape requests specifying either type 3400 tape drive or type 2400 tape drive units with 9-track tape.

- Entry: IEFXV003
- Exit: IEFXV002, to caller
- Tables/Work Areas: ACB, AWT, UCB, LCT
- Attributes: Reusable
- Control Section: IEFXV003

IEFX300A: I/O Device Allocation - Device Strikeout Routine

The purpose of this routine is to modify the primary and secondary bit patterns in AWT entries to complete the allocation to a data set.

- Entry: IEFX3000, X33B42
- Exit: Return to caller
- Tables/Work Areas: AWT, AVT, UCB, LCT
- Control Sections: IEFX3000

IEFX5000: I/O Device Allocation - Decision Allocation Routine

The purpose of this routine is to select devices for data sets with multiple unit possibilities.

- Entry: IEFX5000, XIIB32, X55C86, X55D3G
- Exit: To IEFWCFAK, IEFXJFAK
- Tables/Work Areas: LCT, AWT, AVT, UCB
- Control Sections: IEFX5000

IEFYNIMP: Termination - Step Termination Control Routine

This routine passes control among the modules of the Step Termination routine and, when required, passes control to the Job Termination routine.

- Entry: IEFYN
- Exit: To IEFW22SD, IEFYPJB3, IEFVJIMP, IEFZAJB3
- Tables/Work Areas: JCT, SCT, LCT
- Control Sections: IEFYN

IEFYNMSG: Termination - Step Termination

The purpose of this module is to contain the messages required for the Step Termination Control routine.

- Entry: IEFYNMSG, STRMSG01
- Attributes: Non-executable
- Control Sections: IEFYNMSG

IEFYPJB3: Termination - Step Termination Data Set Driver Routine

The purpose of this routine is to obtain SIOTs and to pass control to the Disposition and Unallocation routine.

- Entry: IEFYP
- Exit: To IEFZGST1, IEFYNIMP

- Tables/Work Areas: LCT, TIOT, UCB, QMPA, SIOT, TCB
- Control Sections: IEFYP

IEFYPMSG: Termination - Step Termination Messages

The purpose of this routine is to contain the messages required by the Step Termination routine.

- Entry: IEFYPMSG, YPPMSG1, YPPMSG2
- Attributes: Non-executable
- Control Sections: IEFYPMSG

IEFYSVMS: Message Blocking Routine

The purpose of this routine is to block system messages into SMBs, and to place SMBs into the message class queue entry.

UCS Universal Character

OLTEP online test executive program

- Entry: IEFYS
- Exit: Return to caller
- Tables/Work Areas: LCT, SCT, SMB
- Attributes: Reentrant
- Control Sections: IEFYS

IEFYTVMS: DSB Processing Routine

The purpose of this routine is to place data set blocks in the space reserved for them in the output queue entries. The routine also marks the DSB inactive if the corresponding SYSOUT entry qualifies for DSO facilities.

- Entry: IEFYT
- Exit: Return to caller
- Tables/Work Areas: JCT, SCT, TIOT, SIOT, QMPA, DSB, JMR, DSOCB
- Attributes: Reentrant
- Control Sections: IEFYT

IEFZAJB3: Termination - Job Termination Control Routine

The purpose of this routine is to provide entry to the Job Termination routine, obtain PDQ blocks, and pass control to the Disposition and Unallocation routine.

- Entry: IEFZA
- Exit: To IEFZGJB1, IEFSD31Q
- Tables/Work Areas: LCT, JCT, PDQ, UCB, QMPA
- Control Sections: IEFZA

IEFZGJB1: Termination - Disposition and Unallocation Routine

The purpose of this routine is to direct the disposition and unallocation of those data sets that remain to be processed at job termination: passed data sets that were not received, and retained data sets that were not referred to.

- Entry: IEFZGJ, ZP0QM
- Exit: Return to caller
- Tables/Work Areas: JCT, PDQ, JFCB, LCT, QMPA, UCB
- Control Sections: IEFZGJ

IEFZGMSG: Termination - Disposition and Unallocation Messages

The purpose of this module is to contain the messages required for the Disposition and Unallocation routine (IEFZGJB1).

- Entry: IEFZGMSG
- Attributes: Non-executable
- Control Sections: IEFZGMSG

IEFZGST1: Termination - Disposition Routine

The purpose of this routine is to direct the disposition of data sets as specified in the DISP field of the DD statement.

- Entry: IEFZG, ZG0J5, ZG0J8
- Exit: To IEFZGST2 or return to caller
- Tables/Work Areas: LCT, PDQ, SIOT, TIOT, UCB, JFCB, QMPA
- Control Sections: IEFZG

IEFZGST2: Termination - Unallocation Routine

The purpose of this routine is to make the units used by this jobstep available for allocation to other data sets.

- Entry: IEFZG2, ZGOK09, ZOOA1, ZOOE10, ZPOC10, ZPOQMGR1

- Exit: Return to caller

- Tables/Work Areas: LCT, PDQ, SIOT, TIOT, UCB, JFCB, QMPA

- Control Sections: IEFZG2

IEFZHMSG: Termination - Disposition and Unallocation Message Routine

The purpose of this routine is to prepare messages to the programmer and to the operator for the Disposition and Allocation routines.

- Entry: IEFZH, ZG0E60, ZK0D1, ZK0E1, XPS631

- Exit: Return to caller

- Tables/Work Areas: LCT, QMPA, SMB, TJB (TSO), TSCVT (TSO)

- Control Sections: IEFZH

IEF060SD: MVT Initiator - Linkage Module

The purpose of this routine is to transfer control to module IEFSD160.

- Entry: IEFSD060

- Exit: To IEFSD160

- Attributes: Reentrant

IEF061SD: Linkage Module

The purpose of this routine is to transfer control to module IEFSD161.

- Entry: IEFSD061

- Exit: To IEFSD161

- Attributes: Reentrant

IEF065SD: Linkage Module

The purpose of this routine is to transfer control to module IEFSD165.

- Entry: IEFSD065

- Exit: To IEFSD165

IEF078SD: Linkage Module

The purpose of this routine is to transfer control to module IEFSD078.

- Entry: IEFSD078

- Exit: To IEFSD078

- Attributes: Reentrant

IEF079SD: Linkage Module

The purpose of this routine is to transfer control to IEFSD079.

- Entry: IEFSD079

- Exit: To IEFSD079

- Attributes: Reentrant

IEF082SD: System Output Writer - Linkage Module

The purpose of this routine is to pass control to the System Output Writer Main Processing routine.

- Entry: IEFSD082

- Exit: To IEFSD082

- Control Sections: IEFSD082

IEF083SD: System Output Writer - Linkage Module

The purpose of this routine is to pass control to the System Output Writer Command Processing routine.

- Entry: IEFSD083

- Exit: IEFSD083

- Control Sections: IEFSD083

IEF300SD: MVT Linkage Module

The purpose of this routine is to provide a linkage to the System Restart Initialization routine.

- Entry: IEFSD300

- Exit: To IEFSD300, IEFSD055

- Attributes: Reentrant

- Control Sections: IEFSD300

IEF304SD: MVT Linkage Module

The purpose of this routine is to provide a linkage to the System Restart Scratch Data Sets routine.

- Entry: IEFSD304

- Exit: To IEFSD304, IEFSD055

- Attributes: Reentrant

- Control Sections: IEFSD304

IEF41FAK: Linkage Module

The purpose of this routine is to provide a linkage to the Allocation Exit routine during graceful step flush.

- Entry: IEFW41SD, IEFW1FAK, IEFW2FAK

- Exit: To IEFW41SD

- Attributes: Read-only, reenterable

- Control Sections: IEFW41SD

IEZDCODE: Interpreter - Record Decompression Routine

The purpose of this routine is to scan a compressed record and decompress it into the original uncompressed record.

- Entry: IEZDCODE

- Exit: Branch on register 14

- Tables/Work Areas: Decompression parameter area

- Attributes: Refreshable, reenterable, character-code independent

- Control Sections: IEZDCODE

IEZNCODE: Interpreter - Record Compression Routine

The purpose of this routine is to scan a record and compress it by replacing each series of occurrences of a selected character with a count field.

- Entry: IEZNCODE

- Exit: Branch on register 14

- Tables/Work Areas: Compression parameter area

- Attributes: Refreshable, reenterable, character-code independent

- Control Sections: IEZNCODE

IFASMFDP: SMF Dump Routine

This routine copies a full SMF data set that is on a direct access volume into a dump data set on tape.

- Entry: IFASMFDP

- Exit: Return to caller

- Attributes: Reentrant

IGC0003D: SEE IEE0003D

IGC0005B: See IEFVSMBR

IGC0008C: See IEESMF8C

IGC00090: See IEFXQM00

IGC0108C: See IEESMFOP

IGC01090: See IEFXQM01

IGC0203E: See IEFWTP00

IGC0208C: See IEESMFAL

IGC02090: See IEFXQM02

IGC0303D: See IEE0303D

IGC0303E: See IEFWTP01

IGC0403D: See IEE0403D

IGC0403E: See IEFWTP02

IGC0503D: See IEE0503D

IGC0603D: See IEE0603D

IGC0703D: See IEE0703D

IGC0803D: See IEE0803D

IGC0903D: See IEE0903D

IGC1103D: See IEE1103D

IGC1403D: See IEE1403D

IGC2203D: See IEEMPCKR

IGC2303D: See IEE2303D

IGC2403D: See IGF2403D, IGF24MPD, IGF34MPD

IGC2503D: See IGF2503D

IGC2603D: See IGF2603D

IGC2903D: See IEE2903D

IGC3103D: See IEE3103D

IGC3202D: See IEE3203D

IGC3303D: See IEE3303D

IGC3403D: See IGF34MPD

IGC3503D: See IEE3503D

IGC3703D: See IEE3703D

IGC3803D: See IEE3803D

IGC4103D: See IEE4103D

IGC4203D: See IEE4203D

IGC4303D: See IEE4303D

IGC4403D: See IEE4403D

IGC4503D: See IEE4503D

IGC4603D: See IEE4603D

IGC4703D: See IEE4703D

IGC4803D: See IEE4803D

IGC4903D: See IEE4903D

IGC5103D: See IEE5103D

IGC5203D: See IEE5203D

IGC5303D: See IEE5303D

IGC5403D: See IEE5403D

IGC5503D: See IEE5503D

IGC5703D: See IEE5703D

IGC6103D: SVC 34 - Display Configuration Matrix Routine (Load 1)

The purpose of this module is to check the syntax of the DISPLAY M command and to display the status of CPU(s) and channel(s), if requested. If status of devices and storage, or either, is requested, control is transferred to the second load of the routine.

- Entry: IGC6103D
- Exit: XCTL to IGC6203D or return to caller
- Attributes: Reentrant
- Control sections: IGC6103D

IGC6203D: SVC 34 - Display Configuration Matrix Routine (Load 2)

The purpose of this module is to display the status of devices and storage, or either.

- Entry: IGC6203D
- Exit: Return to caller
- Attributes: Reentrant
- Control sections: IGC6203D

IGC6503D: See IEE6503D

IGC6603D: See IEE6603D

IGF08501: SVC 34 - Machine Status Control Routine for Model 85, Part 1

This routine is used only witht he Model 85. The purpose of this routine is to process the STATUS parameter in the MODE command.

- Entry: IGF08501
- Exit: IGF08502 or return to caller
- Tables/Work Areas: CVT, XSA, MSB
- Attributes: Reentrant, read-only, self-relocating
- Control Sections: IGF08501

IGF08502: SVC 34 - Machine Status Control Routine for Model 85, Part 2

This routine is used only with the Model 85. The purpose of this routine is to process all parameters of the MODE command except the STATUS parameter.

- Entry: IGF08502
- Exit: Return to caller of SVC 34
- Tables/Work Areas: CVT, XSA, MSB
- Attributes: Reentrant, read-only, self-relocating
- Control Sections: IGF08502

IGF24MPD: SVC 34 - VARY PATH Processor for Multiprocessing Routine (Part 1)

The purpose of this routine is to process an operator's request to vary a path to a device and to cause that path to be logically brought online for use by, or logically removed from, the system. This routine along with module IGF34MPD performs the same function as module IGF2403D in a nonmultiprocessing system.

- Entry: IGF24MPD
- Exit: To IGF34MPD, IEE0503D, or IEE2103D
- Attributes: Reentrant
- Tables/Work Areas: Test channel table, SVRB extended save area
- Control Sections: IGF24MPD

IGF2403D: SVC 34 - VARY PATH Processor Routine

The purpose of this routine is to process an operator's request to vary a path to a device and to cause that path to be logically brought online for use by, or logically removed from, the system. The VARY PATH command can only be used if Alternate Path Retry (APR) is in the system.

- Entry: IGF2403D from IEE3203D
- Exit: To IEE0503D or IEE2103D
- Attributes: Reentrant
- Tables/Work Areas: Test channel table, SVRB extended save area.
- Control Sections: IGF2403D

IGF2603D: SVC 34 - MODE Command Router Routine

The purpose of this routine is to determine the model (85, 155, or 165) for the MODE command and to pass control to the appropriate Machine Status Control routine, which processes the command.

- Entry: IGF2603D
- Exit: XCTL to IGF08501 (for Model 85), IGF29601 (for Model 155), or IGF55301 (for Model 165)
- Tables/Work Areas: XSA, MSB
- Attributes: Reenterable, read-only, self-relocating
- Control Sections: IGF2603D

IGF29601: SVC 34 - Machine Status Control Routine for Model 155

This routine is used only with the Model 155. The purpose of this routine is to process the MODE command.

- Entry: IGF29601
- Exit: Return to caller of SVC 34
- Tables/Work Areas: XSA, MSB, CVT
- Attributes: Serially reusable, read-only, self-relocating
- Control Sections: IGF29601

IGF29701: SVC 34 -- Machine Status Control Routine - Model 145

The purpose of this routine is to process the MODE command for the Model 145.

- Entry: IGF29701
- Exit: Return to issuer of SVC 34
- Tables/Work Areas: CVT, MSB, XSA
- Attributes: Reenterable, read-only, transient
- Control Section: IGF29701

IGF34MPD: SVC 34 - VARY PATH Processor for Multiprocessing Routine (Part 2)

The purpose of this routine is to continue the processing of the VARY PATH command that was begun by module IGF24MPD in a multiprocessing system.

- Entry: IGF34MPD
- Exit: To IEE0503D or IEE2103D
- Attributes: Reentrant
- Tables/Work Areas: Test channel table, SVRB extended save area
- Control Sections: IGF34MPD

IGF55301: SVC 34 - Machine Status Control Routine for Model 165

This routine is used only with the Model 165. The purpose of this routine is to process the MODE command.

- Entry: IGF55301
- Exit: Return to caller of SVC 34
- Tables/Work Areas: XSA, MSB, CVT
- Attributes: Serially reusable, read-only, self-relocating
- Control Section: IGF55301

IGG2103D: See IEE2103D

IKJNULL: Error Conditions Router Routine

This module sets the error code in XSA and a preformatted error message is issued from IEE0503D.

- Entry: IKJNULL
- Exit: IEE0503D
- Tables/Work Area: XSA
- Control Sections: IKJNULL
- Attributes: Reusable

Appendix C: Module Cross-Reference

This appendix consists of five alphabetized lists containing information about MVT job management load modules and assembly modules:

- Appendix C1 is an alphabetized list of load module names. Listed next to each load module name are its aliases (if there are any), the assembly modules contained in the load module, the entry points for the assembly modules, and the control section names used in the assembly modules.
- Appendix C2 is an alphabetized list of alias names for MVT job management load modules. Listed next to each alias name is the load module for which it is the alias, the assembly modules contained in the load module, the entry points for the assembly modules, and the control sections used in the assembly modules.
- Appendix C3 is an alphabetized list of assembly module names. Listed next to each assembly module name are the entry points for the module, the control sections used in the module, the load modules in which it is contained, and the aliases for the load modules.
- Appendix C4 is an alphabetized list of entry point names for MVT job management assembly modules. Listed next to each entry point name are the assembly modules for which it is the entry point, the control sections used in the assembly modules, the load modules in which the assembly modules are contained, and the aliases for the load modules.
- Appendix C5 is an alphabetized list of control section names used in MVT job management assembly modules. Listed next to each control section name are the assembly modules in which it is used, the entry points for the assembly modules, the load modules in which the assembly modules are contained, and the aliases for the load modules.

For further information about the assembly modules, refer to Appendix B and to the appropriate portions of the text.

Load Module Name	Alias	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
IEECVCTI		IEEVRFRX	IEEVRFRX	IEEVRFRX
		IEECVINT	IEECVCTI	IEECVCTI
IEELWAIT		IEELWAIT	IEELWAIT	IEELWAIT
IEEPALTR		IEEPALTR	IEEPALTR	IEEPSTP1
IEEPDISC		IEEPDISC	IEEPDISC	IEEPDISC
IEEPPRES		IEEPPRES	IEEPPRES	IEEPPRES
IEEPRTN		IEEPRTN2	IEEPRTN2	IEEPRTN2
IEEPRWI2		IEEPRWI2	IEEPRWI2	IEEPRWI2
IEESD563		IEESD563	IEESD563	IEESD563
IEESD564		IEESD564	IEESD564	IEESD564
IEESD565		IEESD565	IEESD565	IEESD565
IEESD575	IEESD581	IEESD581	IEESD581	IEESD581
		IEESD575	IEESD575	IEESD575
IEESD576	IEESD580	IEESD576	IEESD576	IEESD576
		IEFQDELQ	IEFQDELE	IEFQDELE
		IEESD580	IEESD580	IEESD580
IEESD577		IEESD577	IEESD577	IEESD577
		IEFQMNQQ	IEFQMNQ2	IEFQMNQ2
IEESD578		IEESD578	IEESD578	IEESD578
IEESD579		IEESD579	IEESD579	IEEMSWTO
				IEESD579
		IEFQMNQQ	IEFQMQ2	IEFQMNQ2
IEESD582	IEESD82A	IEESD582	IEESD582	IEESD582
			IEESD82A	
IEESD583		IEESD583	IEESD583	IEESD583
IEESD584		IEESD584	IEESD584	IEESD584
IEESMFIT	IEESMFI4	IEESMFIT	IEESMFIT	IEESMFIT
			IEESMFI4	
IEESMFI2		IEESMFI2	IEESMFI2	IEESMFI2
IEESMFI3	IEESMFMS	IEESMFI3	IEESMFI0	IEESMFI3
	IEESMFI0		IEESMFI3	
			IEESMFMS	
IEESMFOI		IEESMFOI	IEESMFOI	IEESMFOI
IEESMFWR		IEESMFWR	IEESMFWR	IEESMFWR
IEEVDNX1	IEESD562	IEESD562	IEESD562	IEESD562
IEEVDSP1		IEEVDSP1	IEEVDSP1	IEEVDSP1
		IEEVDRGN	IEEVDRGN	IEEVDRGN
IEEVICLR	IEFICR	IEEVICLR	IEEVICLR	IEEVICLR
			IEFICR	
IEEVICTL	IEEVIC	IEEVICTL	IEEVICTL	IEEVICTL
		IEEVSMSG	IEEVSMSG	IEEVSMSG
IEEVIPL		IEEVIPL	IEEVIPL	IEEVIPL
IEEVLDSP		IEEVLDSP	IEEVLDSP	IEEVLDSP
IEEVLIN		IEEVRFRX	IEEVRFRX	IEEVRFRX
		IEEVLIN1	IEEVLIN1	IEEVLIN1
IEEVLNKT		IEEVLNKT		
IEEVLOUT		IEEVLOUT	IEEVLOUT	IEEVLOUT
IEEVMNT1		IEEVMNT1	IEEVMNT1	IEEVMNT1
		IEEVJCL	IEEVJCL	IEEVJCL
IEEVMNT2		IEEVSMSG	IEEVSMSG	IEEVSMSG
		IEEVMNT2	IEEVMNT2	IEEVMNT2
IEEVOMSG		IEEVOMSG	IEEVOMSG	IEEVOMSG
IEEVPRES		IEFPRES	IEFPRES	IEFPRES
		IEFK1MSG	IEFK1MSG	IEFK1MSG
		IEEVPRES	IEEVPRES	IEEVPRES

(Continued)

Load Module Name	Alias	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
IEEVRCTL		IEEVRCTL	IEEVRCTL	IEEVRCTL
IEEVRJCL	IEEPSN	IEEVRJCL	IEEVRJCL	IEEVRJCL
		IEEPSN	IEEPSN	IEEPSN
		IEEVLNKT		IEEVLNKT
IEEVSIPL		IEEVSIPL	IEEVSIPL	IEEVSIPL
IEEVSTAR		IEEVSTAR	IEEVSTAR	IEEVSTAR
				IEEVJCL
		IEEVJCL	IEEVJCL	IEEVJCL
IEEVWAIT	IEEVWDAR	IEEVWAIT	IEEVWAIT	IEEVWAIT
IEEXEDNA		IEEXEDNA	IEEXEDNA	IEEXEDNA
IEE0503D		IEE0503D	IEE0503D	IEE0503D
IGC5503D		IEE5503D	IEE5503D	IEE5503D
IGC5803D		IKJ5803D	IKJ5803D	IKJ5803D
		IKJNULL	IKJNULL	IKJNULL
IEFBR14		IEFBR14	IEFBR14	IEFBR14
IEFDSDRP		IEFQASGQ	IEFQASGN	IEFQASNM
				IEFQASGN
		IEFDSDRP	IEFDSDRP	IEFDSDRP
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFQBVMS	IEFQMSSS	IEFQMSSS
IEFDSO		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFDSOCP	IEFDSOCP	IEFDSOCP
IEFDSOAL		IEFDSOAL	IEFDSOAL	IEFDSOAL
IEFDSOCR		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFQMNQQ	IEFQMNQ2	IEFQMNQ2
		IEFDSOCR	IEFDSOCR	IEFDSOCR
		IEFSD168	IEFSD068	IEFSD068
		IEFSD112	IEFSD112	IEFSD112
		IEFSD111	IEFSD111	IEFSD111
IEFDSOFB		IEFDSOFB	IEFDSOFB	IEFDSOFB
IEFDSOLP		IEFDSOLP	IEFDSOLP	IEFDSOLP
IEFDSOSM		IEFDSOSM	IEFDSOSM	IEFDSOSM
IEFDSOWR		IEFDSOWR	IEFDSOWR	IEFDSOWR
IEFIIC		IEFIIC	IEFIIC	IEFIIC
IEFIRC		IEFIRC	IEFIRC	IEFIRC
		IEEVSMSG	IEEVSMSG	IEEVSMSG
IEFLOCDQ	LOCDQ	IEFLOCDQ	LOCDQ	IEFLOCDQ
	LOC		LOC	
	LOCCAN		LOCCAN	
		IEFCNVRT	CNVRT	IEFCNVRT
		IEFRDWRT	WRT	IEFRDWRT
			RD	
IEFMCVOL	IEFCVOL3	IEF41FAK	IEFW2FAK	IEFW41SD
	IEFCVOL2		IEFW1FAK	
	IEFCVOL1		IEFW41SD	
		IEFYSVMS	IEFYS	IEFYS
		IEFVMMS1	IEFVM1	IEFVM1
		IEFVMLS7	IEFVM7	IEFVM7
		IEFVMLS6	IEFVMSGR	IEFVM6
		IEFVMFAK	IEFVMCVL	IEFVMCVL
		IEFSD111	IEFSD111	IEFSD111
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFQMLK1	IEFQMSSS	IEFQMSSS

(Continued)

Load Module Name	Alias	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
		IEFQASGQ	IEFQASGN	IEFQASNM
				IEFQASGN
		IEFMCVOL	IEFCVOL3	IEFCVOL3
			IEFCVOL2	IEFCVOL2
			IEFCVOL1	IEFCVOL1
IEFPRINT	IEFPRT	IEFPRTXX	SPRINTER	SPRINTER
IEFQDELE		IEFQDELQ	IEFQDELE	IEFQDELE
IEFQINTZ		IEFQINTZ	IEFQINTZ	IEFQINTZ
IEFQMDQQ	IEFQMDQ2	IEFQMDQQ	IEFQMDQ2	IEFQMDQ2
IEFQMNQ2		IEFQMNQQ	IEFQMNQ2	IEFQMNQ2
IEFQMRAW		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
IEFQMSSS	IEFIRCB	IEFQAGST	IEFQAGST	IEFQAGST
		IEFIRCB	IEFIRCB	IEFIRCB
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFQMNQQ	IEFQMNQ2	IEFQMNQ2
		IEFQBVMS	IEFQMSSS	IEFQMSSS
		IEFQASGQ	IEFQASGN	IEFQASNM
				IEFQASGN
IEFQMUNC		IEFQMUNQ	IEFQMUNC	IEFQMUNC
IEFRCLN1		IEFRCLN1	IEFRCLN1	IEFRCLN1
IEFRCLN2		IEFRCLN2	IEFRCLN2	IEFRCLN2
IEFRSTRT	IEFSMR	IEFRSTRT	IEFSMR	IEFRSTRT
			IEFRSTRT	
IEFSDTTE		IEFSDTTE	IEFSDTTE	IEFSDTTE
IEFSDXXX		IEFSDXXX	IEFSDXXX	IEFSDXXX
IEFSDXYZ		IEFSDXYZ	IEFSDXYZ	IEFSDXYZ
IEFSD0XX		IEFSD0XX	IEFSD0XX	IEFSD0XX
IEFSD060		IEF061SD	IEFSD061	IEFSD061
		IEFSD160	IEFSD060	IEFSD60M
				IEFSD060
IEFSD061	IEFW42SD	IEFQASGQ	IEFQASGN	IEFQASNM
				IEFQASGN
	IEFSD104	IEFIDUMP	IEFIDUMP	IEFIDUMP
	IEFSD064	IEFIDMPM	IEFIDMPM	IEFIDMPM
		IEFDSTRT	IEFDSTRT	IEFDSTRT
		IEFDSTBL	IEFDSTBL	IEFDSTBL
		IEFDSLST	IEFDSLST	IEFDSLST
		IEFACTRT	IEFACTRT	IEFACTRT
		IEFZGST1	ZGOJ8	IEFZG
			ZGOJ5	
			IEFZG	
		IEFYRCDS	IEFYRCDS	IEFSD572
				IEFYRCDS
		IEFYPMSG	YPPMSG2	IEFYPMSG
			YPPMSG1	
			IEFYPMSG	
		IEFYPJB3	IEFYP	IEFYP
		IEFYNMSG	STRMSG01	IEFYNMSG
			IEFYNMSG	
		IEFYNIMP	IEFYN	IEFYN
		IEFWTERM	IEFWTERM	IEFWTERM
		IEFVSDRA	IEFVSDRA	IEFVSDRA
		IEFVJMSG	IEFVJMSG	IEFVJMSG
		IEFVJIMP	IEFVJ	IEFVJ
		IEFUSI	IEFUSI	IEFUSI
		IEFUJI	IEFUJI	IEFUJI
		IEFSMFWI	IEFSMFWI	IEFSMFWI
		IEFSMFLK	IEFACTLK	IEFACTLK
		IEFSMFIE	IEFSMFIE	IEFSMFIE
		IEFSD514	IEFSD514	IEFSD514

(Continued)

Load Module Name	Alias	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
		IEFSD42Q	IEFW42SD	IEFW42SD
		IEFSD31Q	IEFW31SD	IEFW31SD
		IEFSD22Q	IEFW22SD	IEFW22SD
		IEFSD168	IEFSD068	IEFSD068
		IEFSD166	IEFSD066	IEFSD066
		IEFSD164	IEFSD064	IEFSD064
		IEFSD161	IEFSD061	IEFDSOAL
				QMTMSD
				IEFSD061
		IEFSD112	IEFSD112	IEFSD112
		IEFSD111	IEFSD111	IEFSD111
		IEFSD110	IEFSD110	IEFSD110
		IEFSD104	IEFSD104	IEFSD104
		IEFSD101	IEFSD101	IEFSD101
		IEFRPREP	IEFRPREP	IEFRPREP
		IEFQMUNQ	IEFQMUNC	IEFQMUNC
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFQMNQQ	IEFQMNQ2	IEFQMNQ2
		IEFQMLK1	IEFQMSSS	IEFQMSSS
		IEFQMDQQ	IEFQMDQ2	IEFQMDQ2
		IEFQDELQ	IEFQDELE	IEFQDELE
		IEFSDPPT		IEFSDPPT
		IEF060SD	IEFSD060	IEFSD060
		IEFZGJBL	ZP0QM	IEFZGJ
		IEFZGST2	ZPOQMGR1	IEFZG2
			ZPOC10	
			IEFZG2	
			ZGOK09	
			ZOOE10	
			ZOOA1	
		IEFZHMSG	XPS631	IEFZH
			ZK0E1	
			ZK0D1	
			ZG0E60	
			IEFZH	
IEFSD062	IEFV4221	IEFQASGQ	IEFQASGN	IEFQASNM
				IEFQASGN
		IEFSD111	IEFSD111	IEFSD111
		IEFSD110	IEFSD110	IEFSD110
		IEFSD103	IEFSD063	IEFSD063
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEF065SD	IEFSD065	IEFSD065
		IEF061SD	IEFSD061	IEFSD061
		IEFSD514	IEFSD514	IEFSD514
		IEFSD162	IEFSD062	IEFSD062
		IEFSD112	IEFSD112	IEFSD112
IEFSD070		IEFSD070	IEFSD070	IEFSD070
IEFSD078		IEF079SD	IEFSD079	IEFSD079
		IEFSD078	IEFSD078	IEFSD078
IEFSD080	IEFSD079	IEFSD082	IEFSD082	IEFSD082
	IEEVWTR1	IEFSD081	IEFSD081	IEFSD081
		IEFSD080	IEFSD080	IEFSD080
		IEFSD079	IEFSD079	IEFSD079
		IEFQMUNQ	IEFQMUNC	IEFQMUNC
		IEFQMDQQ	IEFQMDQ2	IEFQMDQ2
		IEFQDELQ	IEFQDELE	IEFQDELE
		IEF078SD	IEFSD078	IEFSD078
		IEFSD084	IEFSD084	IEFSD084
		IEFSD083	IEFSD083	IEFSD83M
				IEFSD083

(Continued)

Load Module Name	Alias	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
IEFSD085	IEF850SD	IEFSD171	IEFSD071	IEFSD71M
	IEF085SD			IEFSD071
	IEFSD071	IEFSD085	IEF850SD	IEFSD85M
			IEF085SD	IEFSD085
			IEFSD085	
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
IEFSD086	IEF086SD	IEFSD086	IEFSD086	IEFSD086
			IEF086SD	IEFSD86M
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFSD089	IEFSD089	IEFSD89M
				IEFSD089
		IEFSD088	IEFSD088	IEFSD088
IEFSD087		IEFSD089	IEFSD089	IEFSD89M
				IEFSD089
		IEFSD088	IEFSD088	IEFSD088
		IEFSD087	IEFSD087	IEFSD87M
				IEFSD087
IEFSD094		IEFSD089	IEFSD089	IEFSD89M
				IEFSD089
		IEFSD088	IEFSD088	IEFSD088
		IEFSD095	IEFSD095	IEFSD095
		IEFSD094	IEFSD094	IEFSD094
IEFSD102		IEFSD102	IEFSD102	IEFSD102
IEFSD105		IEFSD105	IEFSD105	IEFSD105
IEFSD263		IEFUTL	IEFUTL	IEFUTL
		IEFUSO	IEFUSO	IEFUSO
		IEFSMFAT	IEFSMFAT	IEFSMFAT
		IEFSD263	IEFSD263	IEFSD263
IEFSD300		IEFSD312	SD305MG1	IEFSD312
			SD304MG2	
			SD304MG1	
			IEFSD312	
		IEFSD310	IEFSD310	IEFSD310
		IEFSD303	IEFSD303	IEFSD303
		IEFSD302	IEFSD302	IEFSD302
		IEFSD301	IEFSD301	IEFSD301
		IEFSD300	IEFSD300	IEFSD300
IEFSD304		IEFQDELQ	IEFQDELE	IEFQDELE
		IEFQASGQ	IEFQASGN	IEFQASNM
				IEFQASGN
		IEFSD305	IEFSD305	IEFSD305
		IEFSD304	IEFSD304	IEFSD304
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFQMNQQ	IEFQMNQ2	IEFQMNQ2
		IEFVSDRD	IEFVSDRD	IEFVSDRD
		IEFVSDRA	IEFVSDRA	IEFVSDRA
		IEFSD514	IEFSD514	IEFSD514
		IEFSD312	SD305MG1	IEFSD312
			SD304MG2	
			SD304MG1	
			IEFSD312	
		IEFSD310	IEFSD310	IEFSD310
IEFSD308		IEFSD308	IEFSD308	IEFSD308
IEFSQINT		IEFSD055	IEFSD055	IEFSD055
		IEFQMSGV		IEFQMSGV
		IEFORMAT	IEFORMAT	IEFORMAT
		IEFSD311	SD55MSG3	IEFSD311
			SD55MSG2	
			SD55MSG1	
			IEFSD311	
		IEF304SD	IEFSD304	IEFSD304

(Continued)

Load Module Name	Alias	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
		IEF300SD	IEFSD300	IEFSD300
IEFUJV		IEFUJV	IEFUJV	IEFUJV
IEFVGM1		IEFVGM1		IEFVGM1
IEFVGM10		IEFVGM10		IEFVGM10
IEFVGM11		IEFVGM11		IEFVGM11
IEFVGM12		IEFVGM12		IEFVGM12
IEFVGM13		IEFVGM13		IEFVGM13
IEFVGM14		IEFVGM14		IEFVGM14
IEFVGM15		IEFVGM15		IEFVGM15
IEFVGM16		IEFVGM16		IEFVGM16
IEFVGM17		IEFVGM17		IEFVGM17
IEFVGM18		IEFVGM18		IEFVGM18
IEFVGM19		IEFVGM19		IEFVGM19
IEFVGM2		IEFVGM2		IEFVGM2
IEFVGM3		IEFVGM3		IEFVGM3
IEFVGM4		IEFVGM4		IEFVGM4
IEFVGM5		IEFVGM5		IEFVGM5
IEFVGM6		IEFVGM6		IEFVGM6
IEFVGM7		IEFVGM7		IEFVGM7
IEFVGM70		IEFVGM70		IEFVGM70
IEFVGM71		IEFVGM71		IEFVGM71
IEFVGM78		IEFVGM78		IEFVGM78
IEFVGM8		IEFVGM8		IEFVGM8
IEFVGM9		IEFVGM9		IEFVGM9
IEFVHA	IEFVHCB	IEFVDA	IEFVDA	IEFVDA
		IEFQMDUM		IEFQMSSS
		IEFACT	IEFACT	IEFACT
		IEFVFB	IEFVFB	IEFVFB
		IEFVFA	IEFVFA	IEFVFA
		IEFVEA	IEFVEA	IEFVEA
		IEFVDBSD	IEFVDBSD	IEFVDBSD
		IEFVSD13	IEFSD090	IEFSD090
		IEFVSD12	IEFSD012	IEFSD012
		IEFVKG	IEFVKG	IEFVKG
		IEFVJA	IEFVJA	IEFVJA
		IEFVIND	IEFVIND	IEFVIND
		IEFVHR	IEFVHR	IEFVHR
		IEFVHQ	IEFVHQ	IEFVHQ
		IEFVHM	IEFVHM	IEFVHM
		IEFVHL	IEFVHL	IEFVHL
		IEFVHHB	IEFVHHB	IEFVHHB
		IEFVHH	IEFVHH	IEFVHH
		IEFVHG	IEFVHG	IEFVHG
		IEFVHF	IEFVHF	IEFVHF
		IEFVHEC	IEFVHEC	IEFVHEC
		IEFVHEB	IEFVHEB	IEFVHEB
		IEFVHE	IEFVHE	IEFVHE
		IEFVHCB	IEFVHCB	IEFVHCB
		IEFVHC	IEFVHC	IEFVHC
		IEFVHB	IEFVHB	IEFVHB
		IEFVHAA	IEFVHAA	IEFVHAA
		IEFVHA	IEFVHA	IEFVHA
		IEFVGT	IEFVGT	IEFVGT
		IEFVGS	IEFVGS	IEFVGS
		IEFVGM	IEFVGM	IEFVGM
		IEFVGK	IEFVGK	IEFVGK
		IEFVGI	IEFVGI	IEFVGI
IEFVHN		IEFVHN	IEFVHN	IEFVHN
IEFVH1		IEFVSMSG	IEFVSMSG	IEFVSMSG
		IEFVH2	IEFVH2	IEFVH2
		IEFVH1	IEFVH1	IEFVH1
IEFVINA		IEZNCODE	IEZNCODE	IEZNCODE
		IEZDCODE	IEZDCODE	IEZDCODE
		IEFVINE	IEFVINE	IEFVINE

(Continued)

Load Module Name	Alias	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
		IEFVINC	IEFVINC	IEFVINC
		IEFVINB	IEFVINB	IEFVINB
		IEFVINA	IEFVINA	IEFVINA
		IEFVHR	IEFVHR	IEFVHR
		IEFVHQ	IEFVHQ	IEFVHQ
		IEFVGM	IEFVGM	IEFVGM
IEFVINB		IEFVINB	IEFVINB	IEFVINB
IEFVMA		IEFVMA	IEFVMA	IEFVMA
IEFVMB		IEZNCODE	IEZNCODE	IEZNCODE
		IEFVMB	IEFVMB	IEFVMB
		IEFQMNQQ	IEFQMNQ2	IEFQMNQ2
		IEFQASGQ	IEFQASGN	IEFQASNM
				IEFQASGN
		IEFQAGST	IEFQAGST	IEFQAGST
IEFVMC		IEFVMC	IEFVMC	IEFVMC
IEFVMD		IEFQDELQ	IEFQDELE	IEFQDELE
		IEFVMD	IEFVMD	IEFVMD
IEFVME		IEFVME	IEFVME	IEFVME
IEFVMF		IEFQASGQ	IEFQASGN	IEFQASGN
				IEFQASNM
		IEFQAGST	IEFQAGST	IEFQAGST
		IEFVMG	IEFVMG	IEFVMG
		IEFVMF	IEFVMF	IEFVMF
		IEFSD112	IEFSD112	IEFSD112
		IEFSD111	IEFSD111	IEFSD111
		IEFSD110	IEFSD110	IEFSD110
		IEFQMUNQ	IEFQMUNC	IEFQMUNC
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFQMNQQ	IEFQMNQ2	IEFQMNQ2
		IEFQMDQQ	IEFQMDQ2	IEFQMDQ2
		IEFQDELQ	IEFQDELE	IEFQDELE
		IEFQBVMS	IEFQMSSS	IEFQMSSS
		IEZDCODE	IEZDCODE	IEZDCODE
		IEFVMH	IEFVMH	IEFVMH
IEFVM6LS	IEFXJX5A	IEFCVFAK	IEFCVOL2	IEFCVOL1
	IEFV15XL		IEFCVOL1	
	IEFXJ000		IEFCVOL3	
	IEFXK000	IEFVMLS6	IEFVMSGR	IEFVM6
		IEFSD41Q	IEFW2FAK	IEFW41SD
			IEFW1FAK	
			IEFW41SD	
		IEFSD195	IEFVAWAT	IEFSD095
		IEFSD111	IEFSD111	IEFSD111
		IEFSD096	IEFSD096	IEFSD096
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFQMLK1	IEFQMSSS	IEFQMSSS
		IEFQASGQ	IEFQASGN	IEFQASNM
				IEFQASGN
		IEFYSVMS	IEFYS	IEFYS
		IEFXKMSG	IEFXKMSG	IEFXKMSG
		IEFXKIMP	IEFXK000	IEFXK000
		IEFXJMSG	MSOFF	IEFXJMSG
			MSSYS	
			MSRCV	
		IEFXJIMP	IEFXJX5A	IEFXJ000
			IEFV15XL	
			IEFXJ000	
		IEFXAFAK	IEFXAFAK	IEFXA
		IEFVMMS1	IEFVM1	IEFVM1
		IEFVMLS7	IEFVM7	IEFVM7
IEFVRRC	IEFVRRCB	IEFSD514	IEFSD514	IEFSD514
	IEFVRRCA	IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFQASGQ	IEFQASGN	IEFQASNM
				IEFQASGN
		IEFVRRC	IEFVRRCB	IEFVRRC
			IEFVRRCA	
			IEFVRRC	
IEFVRR1		IEFCNVRT	CNVRT	IEFCNVRT
		IEFVRR1	IEFVRR1	IEFVRR1
		IEFRDWRT	WRT	IEFRDWRT
			RD	
		IEFLOCDQ	LOC	IEFLOCDQ
			LOCCAN	
			LOCDQ	
IEFVRR2	IEFVR2AE	IEFVRR2	IEFV2AE	IEFVRR2
			IEFVRR2	
IEFVRR3	IEFVR3AE	IEFVRR3	IEFV3AE	IEFVRR3
			IEFVRR3	
		IEFSD112	IEFSD112	IEFSD112
		IEFQDELQ	IEFQDELE	IEFQDELE
IEFWA000	IEFWC000	IEFSD552	IEFSD552	IEFXJX5A
		IEFAVFAK		IEFXV001
		IEFWA000	IEFUCBL	IEFWA7
			IEFWA000	IEFWA002
		IEFX300A	X33B42	IEFX3000
			IEFX3000	
		IEFWSWIN	IEFWSWIT	IEFWSWIT
		IEFSGOPT	IEFSGOPT	IEFSGOPT
		IEFSCAN	IEFSCAN	IEFSCAN
			UCBSCAN	
			UCBCONT	
		IEFDEVPT	IEFDEVPT	IEFDEVPT
		IEFXDPTH	IEFXDPTH	IEFXDPTH
		IEFX5000	IEFX5000	IEFX5000
			X55D3G	
			X55C86	
			XIIB32	
		IEFWCIMP	IEFWC000	IEFWC000
		IEFXH000	IEFXH000	IEFXH000
		IEFWDFAK	IEFWD000	IEFWD000
		IEFXJFAK	IEFXJ000	IEFXJ000
		IEF41FAK	IEFW2FAK	IEFW41SD
			IEFW1FAK	
			IEFW41SD	
		IEFSD551	IEFSD551	IEFV15XL
IEFWD000	IEFW41SD	IEFWD000	IEFWD000	IEFWD000
				IEFWD002
				IEFWDMSG
		IEFWD001	IEFWD001	IEFWD001
		IEFWEXTA	IEFWEXTA	IEFWEXTA
		IEFXT00D	WDEXIT	IEFXT000
			XTTBC0	
			IEFXT000	
		IEFXTDMY	IEFXTDMY	IEFXTDMY
		IEFSD195	IEFVAWAT	IEFSD095
		IEFSD096	IEFSD096	IEFSD096
		IEFXT002	IEFXT002	IEFXT002
			XTTEA0	
			XTTEA1	
			XTTEB3	
			XTTRDJ	
		IEFSD097	IEFSD097	IEFSD097
		IEFQMLK1	IEFQMSSS	IEFQMSSS
		IEFYSVMS	IEFYS	IEFYS
		IEFQMRAW	IEFQMRAW	IEFQMRAW

(Continued)

Load Module Name	Alias	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
				IEFQMWTO
		IEFXT003	IEFXT003	IEFXT003
			XUUB00	
			XUUH06	
		IEFXKFAK	IEFXKFAK	IEFXK000
		IEFQASGQ	IEFQASGN	IEFQASGN
				IEFQASNM
		IEFSD41Q	IEFW2FAK	IEFW41SD
			IEFW1FAK	
			IEFW41SD	
		IEFXJFAK	IEFXJ000	IEFXJ000
		IEFVMMS1	IEFVM1	IEFVM1
		IEFSD111	IEFSD111	IEFSD111
				CHFILDRW
		IEFXTMSG	RESMSG	IEFXTMSG
			MSGDSDV	
			MESGHEAD	
			MESGDDEV	
			PRIAMSG	
			JOBCAN	
			WTORREP	
			RESUNIT	
		IEFCVFAK	IEFCVOL1	IEFCVOL1
			IEFCVOL2	
			IEFCVOL3	
IEFWSYP3	IEFWSYP3	IEFWSYP3		
	IEFWSMSG	IEFSWMSG		
IEFW21SD	IEFVMCVL	IEFVKIMP	IEFVK	IEFVK
		IEFSD21Q	IEFW21SD	IEFW21SD
		IEFSD180	IEFSD18	IEFSD180
		IEFSD111	IEFSD111	IEFSD111
				CHFILDRW
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFQMLK1	IEFQMSSS	IEFQMSSS
		IEFQASGQ	IEFQASGN	IEFQASNM
				IEFQASGN
		IEFCVFAK	IEFCVOL3	IEFCVOL1
			IEFCVOL2	
			IEFCVOL1	
		IEFACTFK	IEFACTLK	IEFACTLK
		CHLOADTB		CHLOADTB
		IEF41FAK	IEFW2FAK	IEFW41SD
			IEFW1FAK	
			IEFW41SD	
		IEFYSVMS	IEFYS	IEFYS
		IEFXCSSS	IEFXA	IEFXAB00
				IEFXA
		IEFXAMSG	IEFXAMSG	IEFXAMSG
		IEFWSTRT	IEFWSTRT	IEFWSTRT
		IEFWDFAK	IEFWD000	IEFWD000
		IEFWCFAK	IEFWC000	IEFWC000
		IEFVM76	VM7600	IEFVM76
		IEFVM5LS	VM7300	IEFVM5
		IEFVM4LS	VM7200	IEFVM4
		IEFVM3LS	VM7150	IEFVM3
		IEFVM2LS	VM7100	IEFVM2
		IEFVMLS1	VM7950	IEFVM1
			VM7900	IEFVMQMI
			VM7850	IEFVMPDQ
			VM7750	IEFVMVTE
			VM7742	
			VM7700	
			VM7370	

(Continued)

Load Module Name	Alias	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
			VM7130	
			VM7090	
			VM7070	
			VM7060	
			VM7055AA	
			VM7055	
			VM7000	
			IEFVMQMI	
			IEFVMCVL	
			IEFVM	
		IEFVMLK5	IEFVMLK5	IEFVM6
		IEFVKMSG	IEFVKMJ1	IEFVKMSG
		IEFACTRT	IEFACTRT	IEFACTRT
		IEFWAD	IEFWAD	IEFWAD
				IEFSD572
		IEFWAFAK		IEFWA000
		IEFSCAN	IEFSCAN	IEFSCAN
			UCBSCAN	
			UCBCONT	
		IEFDEVPT	IEFDEVPT	IEFDEVPT
IEFXVAVR	IEFXV001	IEFQMLK1	IEFQMSSS	IEFQMSSS
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFCVFAK	IEFCVOL3	IEFCVOL1
			IEFCVOL2	
			IEFCVOL1	
		IEFX5000	IEFX5000	IEFX5000
			X55D3G	
			X55C86	
			XIIB32	
		IEFX300A	X33B42	IEFX3000
			IEFX3000	
		IEFXV001	IEFXV001	IEFXV001
		IEFXV002	IEFXV002	IEFXV002
		IEFXV003	IEFXV003	IEFXV003
		IEFXVNSL	IEFXVNSL	IEFXVNSL
		IEFXVMSG	IEFXVMSG	IEFXVMSG
		IEFXJFAK	IEFXJ000	IEFXJ000
		IEFXH000	IEFXH000	IEFXH000
		IEFWD001	IEFWD001	IEFWD001
		IEFWD000	IEFWD000	IEFWD000
		IEFWCFAK	IEFWC000	IEFWC000
		IEFSD552	IEFSD552	IEFXJX5A
		IEFSD551	IEFSD551	IEFV15XL
		IEFSD111	IEFSD111	IEFSD111
IGC0003D		IEE0003D	IEE0003D	IEE0003D
IGC0005B		IEFVSMBR	IGC0005B	IGC0005B
IGC0008C		IEESMF8C	IEESMF8C	IGC0083
IGC00110		IEE00110	IEE00110	IEE00110
IGC0108C		IEESMFOP	IEESMFOP	IEESMFOP
IGC0203E		IEFWTP00	IGC0203E	IGC0203E
IGC0208C		IEESMFAL	IEESMFAL	IEESMFAL
IGC0303D		IEE0303D	IEE0303D	IEE0303D
IGC0303E		IEFWTP01	IGC0303E	IGC0303E
IGC0403D		IEE0403D	IEE0403D	IEE0403D
IGC0403E		IEFWTP02	IGC0403E	IGC0403E
IGC0503D		IEE0503D	IEE0503D	IEE0503D
IGC0603D		IEE0603D	IEE0603D	IEE0603D
IGC0703D		IEE0703D	IEE0703D	IEE0703D
IGC0803D		IEE0803D	IEE0803D	IEE0803D
IGC0903D		IEE0903D	IEE0903D	IEE1103D
IGC1B03D		IEE1B03D	IEE1B03D	IEE1B03D
IGC10110		IEE10110	IEE10110	IEE10110
IGC1103D		IEE1103D	IEE1103D	IEE1103D

(Continued)

Load Module Name	Alias	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
IGC11110		IEE11110	IEE11110	IEE11110
IGC1203D		IEE1A03D	IEE1A03D	IEE1A03D
		IEE1203D	IEE1203D	IEE1203D
IGC12110		IEE12110	IEE12110	IEE12110
IGC1403D		IEE1403D	IEE1403D	IEE1403D
IGC1603D		IEE1603D	IEE1603D	IEE1603D
IGC20110		IEE20110	IEE20110	IEE20110
IGC21110		IEE21110	IEE21110	IEE21110
IGC22110		IEE22110	IEE22110	IEE22110
IGC2303D		IEE2303D	IEE2303D	IEE2303D
IGC23110		IEE23110	IEE23110	IEE23110
IGC2903D		IEE2903D	IEE2903D	IEE2903D
IGC30110		IEE30110	IEE30110	IEE30110
IGC3103D		IEE3103D	IEE3103D	IEE3103D
IGC31110		IEE31110	IEE31110	IEE31110
IGC3203D		IEE3203D	IEE3203D	IEE3203D
IGC32110		IEE32110	IEE32110	IEE32110
IGC3303D		IEE3303D	IEE3303D	IEE3303D
IGC3503D		IEE3503D	IEE3503D	IEE3503D
IGC3603D		IEE3603D	IEE3603D	IEE3603D
IGC3703D		IEE3703D	IEE3703D	IEE3703D
IGC3803D		IEE3803D	IEE3803D	IEE3803D
IGC40110		IEE40110	IEE40110	IEE40110
IGC4103D		IEE4103D	IEE4103D	IEE4103D
IGC4203D		IEE4203D	IEE4203D	IEE4203D
IGC4303D		IEE4303D	IEE4303D	IEE4303D
IGC4403D		IEE4403D	IEE4403D	IEE4403D
IGC4503D		IEE4503D	IEE4503D	IEE4503D
IGC4603D		IEE4603D	IEE4603D	IEE4603D
IGC4703D		IEE4703D	IEE4703D	IEE4703D
IGC4803D		IEE4803D	IEE4803D	IEE4803D
IGC4903D		IEE4903D	IEE4903D	IEE4903D
IGC50110		IEE50110	IEE50110	IEE50110
IGC5103D		IEE5103D	IEE5103D	IEE5103D
IGC5203D		IEE5203D	IEE5203D	IEE5203D
IGC5303D		IEE5303D	IEE5303D	IEE5303D
IGC5403D		IEE5403D	IEE5403D	IEE5403D
IGC5603D		IEE5603D	IEE5603D	IEE5603D
IGC5903D		IEE5903D	IEE5903D	IEE5903D
IGC60110		IEE60110	IEE60110	IEE60110
IGC6103D		IGC6103D	IGC6103D	IGC6103D
		IGC6203D	IGC6203D	IGC6203D
IGC6303D		IEE6303D	IEE6303D	IEE6303D
IGC6403D		IEE6403D	IEE6403D	IEE6403D
IGC6503D		IEE6503D	IEE6503D	IEE6503D
IGC6603D		IEE6603D	IEE6603D	IEE6603D
IGC6703D		IEE6703D	IEE6703D	IEE6703D
IGC6803D		IEE6803D	IEE6803D	IEE6803D
IGC6903D		IEE6903D	IEE6903D	IEE6903D
IGC7103D		IEE7103D	IEE7103D	IEE7103D
IGC7203D		IEE7203D	IEE7203D	IEE7203D
IGC7303D		IEE7303D	IEE7303D	IEE7303D
IGC7503D		IEE7503D	IEE7503D	IEE7503D
IGC7603D		IEE7603D	IEE7603D	IEE7603D
IGC7703D		IEE7703D	IEE7703D	IEE7703D
IGC7803D		IEE7803D	IEE7803D	IEE7803D
IGC7903D		IEE7903D	IEE7903D	IEE7903D
IGC8503D		IEE8503D	IEE8503D	IEE8503D
IGC8603D		IEE8603D	IEE8603D	IEE8603D
IGF08501		IGF08501	IGF08501	IGF08501
IGF08502		IGF08502	IGF08502	IGF08502
IGF2603D		IGF2603D	IGF2603D	IGF2603D
IGF29601		IGF29601	IGF29601	IGF29601
IGF29701		IGF29701	IGF29701	IGF29701
IGF55301		IGF55301	IGF55301	IGF55301
IGG2103D		IEE2103D	IEE2103D	IEE2103D

(Continued)

Alias Name	Load Module Name	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
IEEPSN	IEEVRJCL	IEEVRJCL	IEEVRJCL	IEEVRJCL
		IEEPSN	IEEPSN	IEEPSN
		IEEVLNKT		IEEVLNKT
IEESD562	IEEVDNX1	IEESD562	IEESD562	IEESD562
IEESD580	IEESD576	IEFQDELQ	IEFQDELE	IEFQDELE
		IEESD580	IEESD580	IEESD580
		IEESD576	IEESD576	IEESD576
IEESD581	IEESD575	IEESD581	IEESD581	IEESD581
		IEESD575	IEESD575	IEESD575
IEESD82A	IEESD582	IEESD582	IEESD582	IEESD582
			IEESD82A	
IEESMFI0	IEESMFI3	IEESMFI3	IEESMFI3	IEESMFI3
			IEESMFIO	
			IEESMFMS	
IEESMFI4	IEESMFIT	IEESMFIT	IEESMFI4	IEESMFIT
			IEESMFIT	
IEESMFMS	IEESMFI3	IEESMFI3	IEESMFI0	IEESMFI3
			IEESMFI3	
			IEESMFMS	
IEEVIC	IEEVICTL	IEEVICTL	IEEVICTL	IEEVICTL
IEEVWDAR	IEEVWAIT	IEEVWAIT	IEEVWAIT	IEEVWAIT
IEEVWTR1	IEFSD080	IEFQDELQ	IEFQDELE	IEFQDELE
		IEFQMDQQ	IEFQMDQ2	IEFQMDQ2
		IEFQMUNQ	IEFQMUNC	IEFQMUNC
		IEFSD079	IEFSD079	IEFSD079
		IEFSD080	IEFSD080	IEFSD080
		IEFSD081	IEFSD081	IEFSD081
		IEFSD082	IEFSD082	IEFSD082
		IEFSD083	IEFSD083	IEFSD083
				IEFSD83M
		IEFSD084	IEFSD084	IEFSD084
		IEF078SD	IEFSD078	IEFSD078
IEFCVOL1	IEFMCVOL	IEFMCVOL	IEFCVOL1	IEFCVOL1
			IEFCVOL2	IEFCVOL2
			IEFCVOL3	IEFCVOL3
		IEFQASGQ	IEFQASGN	IEFQASGN
				IEFQASNM
		IEF41FAK	IEFW2FAK	IEFW41SD
			IEFW1FAK	
			IEFW41SD	
		IEFYSVMS	IEFYS	IEFYS
		IEFVMMS1	IEFVM1	IEFVM1
		IEFVMLS7	IEFVM7	IEFVM7
		IEFVMLS6	IEFVMSGR	IEFVM6
		IEFVMFAK	IEFVMCVL	IEFVMCVL
		IEFSD111	IEFSD111	IEFSD111
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFQMLK1	IEFQMSSS	IEFQMSSS
IEFCVOL2	IEFMCVOL	IEFMCVOL	IEFCVOL1	IEFCVOL1
			IEFCVOL2	IEFCVOL2
			IEFCVOL3	IEFCVOL3
		IEFQASGQ	IEFQASGN	IEFQASGN
				IEFQASNM
		IEF41FAK	IEFW2FAK	IEFW41SD
			IEFW1FAK	
			IEFW41SD	
		IEFYSVMS	IEFYS	IEFYS

(Continued)

Alias Name	Load Module Name	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
		IEFVMMS1	IEFVM1	IEFVM1
		IEFVMLS7	IEFVM7	IEFVM7
		IEFVMLS6	IEFVMSGR	IEFVM6
		IEFVMFAK	IEFVMCVL	IEFVMCVL
		IEFSD111	IEFSD111	IEFSD111
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFQMLK1	IEFQMSSS	IEFQMSSS
IEFCVOL3	IEFMCVOL	IEFMCVOL	IEFCVOL1	IEFCVOL1
			IEFCVOL2	IEFCVOL2
			IEFCVOL3	IEFCVOL3
		IEFQASGQ	IEFQASGN	IEFQASGN
				IEFQASNM
		IEF41FAK	IEFW2FAK	IEFW41SD
			IEFW1FAK	
			IEFW41SD	
		IEFYSVMS	IEFYS	IEFYS
		IEFVMMS1	IEFVM1	IEFVM1
		IEFVMLS7	IEFVM7	IEFVM7
		IEFVMLS6	IEFVMSGR	IEFVM6
		IEFVMFAK	IEFVMCVL	IEFVMCVL
		IEFSD111	IEFSD111	IEFSD111
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
IEFICR	IEEVICLR	IEEVICLR	IEFICR	IEEVICLR
			IEEVICLR	
IEFIRCB	IEFQMSSS	IEFQAGST	IEFQAGST	IEFQAGST
		IEFIRCB	IEFIRCB	IEFIRCB
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFQMNQQ	IEFQMNQ2	IEFQMNQ2
		IEFQBVMS	IEFQMSSS	IEFQMSSS
		IEFQASGQ	IEFQASGN	IEFQASNM
				IEFQASGN
IEFPRT	IEFPRINT	IEFPRTXX	SPRINTER	SPRINTER
IEFQMDQ2	IEFQMDQQ	IEFQMDQQ	IEFQMDQ2	IEFQMDQ2
IEFSD064	IEFSD061	IEFACTRT	IEFACTRT	IEFACTRT
		IEFDSLST	IEFDSLST	IEFDSLST
		IEFDSTBL	IEFDSTBL	IEFDSTBL
		IEFDSTRT	IEFDSTRT	IEFDSTRT
		IEFIDMPM	IEFIDMPM	IEFIDMPM
		IEFIDUMP	IEFIDUMP	IEFIDUMP
		IEFQASGQ	IEFQASGN	IEFQASGN
				IEFQASNM
		IEFQDELQ	IEFQDELE	IEFQDELE
		IEFQMDQQ	IEFQMDQ2	IEFQMDQ2
		IEFQMLK1	IEFQMSSS	IEFQMSSS
		IEFQMNQQ	IEFQMNQ2	IEFQMNQ2
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFQMUNQ	IEFQMUNC	IEFQMUNC
		IEFRPREP	IEFRPREP	IEFRPREP
		IEFSD101	IEFSD101	IEFSD101
		IEFSD104	IEFSD104	IEFSD104
		IEFSD110	IEFSD110	IEFSD110
		IEFSD111	IEFSD111	IEFSD111
		IEFSD112	IEFSD112	IEFSD112
		IEFSD161	IEFSD061	IEFSD061

(Continued)

Alias Name	Load Module Name	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
				QMTMSD
				IEFDSOAL
		IEFSD165	IEFSD065	IEFSD065
		IEFSD166	IEFSD066	IEFSD066
		IEFSD168	IEFSD068	IEFSD068
		IEFSDPPT		IEFSDPPT
		IEFSD22Q	IEFW22SD	IEFW22SD
		IEFSD31Q	IEFW31SD	IEFW31SD
		IEFSD42Q	IEFW42SD	IEFW42SD
		IEFSD514	IEFSD514	IEFSD514
		IEFSMFIE	IEFSMFIE	IEFSMFIE
		IEFSMFLK	IEFACTLK	IEFACTLK
		IEFSMFWI	IEFSMFWI	IEFSMFWI
		IEFUJI	IEFUJI	IEFUJI
		IEFUSI	IEFUSI	IEFUSI
		IEFVJIMP	IEFVJ	IEFVJ
		IEFVJMSG	IEFVJMSG	IEFVJMSG
		IEFVSDRA	IEFVSDRA	IEFVSDRA
		IEFWTERM	IEFWTERM	IEFWTERM
		IEFYNIMP	IEFYN	IEFYN
		IEFYNMSG	IEFYNMSG	IEFYNMSG
			STRMSG01	
		IEFYPJB3	IEFYP	IEFYP
		IEFYPMSG	IEFYPMSG	IEFYPMSG
			YPPMSG1	
			YPPMSG2	
		IEFYRCDS	IEFYRCDS	IEFYRCDS
				IEFSD572
		IEFZGST1	IEFZG	IEFZG
			ZG0J5	
			ZG0J8	
		IEFZGST2	IEFZG2	IEFZG2
			ZG0K09	
			ZOOA1	
			ZOOE10	
			ZPOC10	
			ZPOQMGR1	
		IEFZHMSG	IEFZH	IEFZH
			ZG0E60	
			ZK0D1	
			ZK0E1	
			XPS631	
		IEF060SD	IEFSD060	IEFSD060
IEFSD071	IEFSD085	IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFSD085	IEFSD085	IEFSD085
			IEF085SD	IEFSD85M
			IEF850SD	
		IEFSD171	IEFSD071	IEFSD071
				IEFSD71M
IEFSD079	IEFSD080	IEF078SD	IEFSD078	IEFSD078
		IEFSD084	IEFSD084	IEFSD084
		IEFSD083	IEFSD083	IEFSD83M
				IEFSD083
		IEFQMUNQ	IEFQMUNC	IEFQMUNC
		IEFQMDQQ	IEFQMDQ2	IEFQMDQ2
		IEFQDELQ	IEFQDELE	IEFQDELE
		IEFSD082	IEFSD082	IEFSD082
		IEFSD081	IEFSD081	IEFSD081
		IEFSD080	IEFSD080	IEFSD080
		IEFSD079	IEFSD079	IEFSD079
IEFSD104	IEFSD061	IEFDSLST	IEFDSLST	IEFDSLST
		IEFDSTBL	IEFDSTBL	IEFDSTBL

(Continued)

Alias Name	Load Module Name	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
		IEFDSTRT	IEFDSTRT	IEFDSTRT
		IEFIDMPM	IEFIDMPM	IEFIDMPM
		IEFSDPPT		IEFSDPPT
		IEFIDUMP	IEFIDUMP	IEFIDUMP
		IEFQASGQ	IEFQASGN	IEFQASGN
				IEFQASNM
		IEFQDELQ	IEFQDELE	IEFQDELE
		IEFQMDQQ	IEFQMDQ2	IEFQMDQ2
		IEFQMLK1	IEFQMSSS	IEFQMSSS
		IEFQMNQQ	IEFQMNQ2	IEFQMNQ2
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFQMUNQ	IEFQMUNC	IEFQMUNC
		IEFRPREP	IEFRPREP	IEFRPREP
		IEFSD101	IEFSD101	IEFSD101
		IEFSD104	IEFSD104	IEFSD104
		IEFSD110	IEFSD110	IEFSD110
		IEFSD111	IEFSD111	IEFSD111
		IEFSD112	IEFSD112	IEFSD112
		IEFSD161	IEFSD061	IEFSD061
				QMTMSD
				IEFDSOAL
		IEFSD164	IEFSD064	IEFSD064
		IEFSD165	IEFSD065	IEFSD065
		IEFSD166	IEFSD066	IEFSD066
		IEFSD168	IEFSD068	IEFSD068
		IEFSD22Q	IEFW22SD	IEFW22SD
		IEFSD31Q	IEFW31SD	IEFW31SD
		IEFSD42Q	IEFW42SD	IEFW42SD
		IEFSD514	IEFSD514	IEFSD514
		IEFSMFIE	IEFSMFIE	IEFSMFIE
		IEFSMFLK	IEFACTLK	IEFACTLK
		IEFSMFWI	IEFSMFWI	IEFSMFWI
		IEFUJI	IEFUJI	IEFUJI
		IEFUSI	IEFUSI	IEFUSI
		IEFVJIMP	IEFVJ	IEFVJ
		IEFVJMSG	IEFVJMSG	IEFVJMSG
		IEFVSDRA	IEFVSDRA	IEFVSDRA
		IEFWTERM	IEFWTERM	IEFWTERM
		IEFYNIMP	IEFYN	IEFYN
		IEFYNMSG	IEFYNMSG	IEFYNMSG
			STRMSG01	
		IEFYPJB3	IEFYP	IEFYP
		IEFYPMSG	IEFYPMSG	IEFYPMSG
			YPPMSG1	
			YPPMSG2	
		IEFYRCDS	IEFYRCDS	IEFYRCDS
				IEFSD572
		IEFZGST1	IEFZG	IEFZG
			ZG0J5	
			ZG0J8	
		IEFZGST2	IEFZG2	IEFZG2
			ZG0K09	
			ZOOA1	
			ZOOE10	
			ZPOC10	
			ZPOQMGR1	
		IEFZHMSG	IEFZH	IEFZH
			ZG0E60	
			ZK0D1	
			ZK0E1	
			XPS631	
		IEF060SD	IEFSD060	IEFSD060
		IEFACTRT	IEFACTRT	IEFACTRT

(Continued)

Alias Name	Load Module Name	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
IEFV4221	IEFSD062	IEF065SD	IEFSD065	IEFSD065
		IEF061SD	IEFSD061	IEFSD061
		IEFSD514	IEFSD514	IEFSD514
		IEFSD162	IEFSD062	IEFSD062
		IEFSD112	IEFSD112	IEFSD112
		IEFSD111	IEFSD111	IEFSD111
		IEFSD110	IEFSD110	IEFSD110
		IEFSD103	IEFSD063	IEFSD063
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFQASGQ	IEFQASGN	IEFQASNM
				IEFQASGN
IEFSMR	IEFRSTRT	IEFRSTRT	IEFSMR	IEFRSTRT
			IEFRSTRT	
IEFVHCB	IEFVHA	IEFVDA	IEFVDA	IEFVDA
		IEFQMDUM		IEFQMSSS
		IEFACT	IEFACT	IEFACT
		IEFVFB	IEFVFB	IEFVFB
		IEFVFA	IEFVFA	IEFVFA
		IEFVEA	IEFVEA	IEFVEA
		IEFVDBSD	IEFVDBSD	IEFVDBSD
		IEFVSD13	IEFSD090	IEFSD090
		IEFVSD12	IEFSD012	IEFSD012
		IEFVKG	IEFVKG	IEFVKG
		IEFVJA	IEFVJA	IEFVJA
		IEFVIND	IEFVIND	IEFVIND
		IEFVHR	IEFVHR	IEFVHR
		IEFVHQ	IEFVHQ	IEFVHQ
		IEFVHM	IEFVHM	IEFVHM
		IEFVHL	IEFVHL	IEFVHL
		IEFVHHB	IEFVHHB	IEFVHHB
		IEFVHH	IEFVHH	IEFVHH
		IEFVHG	IEFVHG	IEFVHG
		IEFVHF	IEFVHF	IEFVHF
		IEFVHEC	IEFVHEC	IEFVHEC
		IEFVHEB	IEFVHEB	IEFVHEB
		IEFVHE	IEFVHE	IEFVHE
		IEFVHCB	IEFVHCB	IEFVHCB
		IEFVHC	IEFVHC	IEFVHC
		IEFVHB	IEFVHB	IEFVHB
		IEFVHAA	IEFVHAA	IEFVHAA
		IEFVHA	IEFVHA	IEFVHA
		IEFVGT	IEFVGT	IEFVGT
		IEFVGS	IEFVGS	IEFVGS
		IEFVGM	IEFVGM	IEFVGM
		IEFVGK	IEFVGK	IEFVGK
		IEFVGI	IEFVGI	IEFVGI
IEFVMCVL	IEFW21SD	IEFVKIMP	IEFVK	IEFVK
		CHLOADTB		CHLOADTB
		IEFACTFK	IEFACTLK	IEFACTLK
		IEFSD21Q	IEFW21SD	IEFW21SD
		IEFSD180	IEFSD18	IEFSD180
		IEFSD111	IEFSD111	IEFSD111
				CHF1LDRW
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFQMLK1	IEFQMSSS	IEFQMSSS
		IEFQASGQ	IEFQASGN	IEFQASNM
				IEFQASGN
		IEFCVFAK	IEFCVOL3	IEFCVOL1
			IEFCVOL2	
			IEFCVOL1	
		IEF41FAK	IEFW2FAK	IEFW41SD
			IEFW1FAK	

(Continued)

Alias Name	Load Module Name	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
			IEFW41SD	
		IEFYSVMS	IEFYS	IEFYS
		IEFACTRT	IEFACTRT	IEFACTRT
		IEFWAD	IEFWAD	IEFWAD
				IEFSD572
		IEFWAFAK		IEFWA000
		IEFSCAN	IEFSCAN	IEFSCAN
			UCBSCAN	
			UCBCONT	
		IEFDEVPT	IEFDEVPT	IEFDEVPT
		IEFXCSSS	IEFXA	IEXAB00
				IEFXA
		IEFXAMSG	IEFXAMSG	IEFXAMSG
		IEFWSTRT	IEFWSTRT	IEFWSTRT
		IEFWDFAK	IEFWD000	IEFWD000
		IEFWCFAK	IEFWC000	IEFWC000
		IEFVM76	VM7600	IEFVM76
		IEFVM5LS	VM7300	IEFVM5
		IEFVM4LS	VM7200	IEFVM4
		IEFVM3LS	VM7150	IEFVM3
		IEFVM2LS	VM7100	IEFVM2
		IEFVMLS1	VM7950	IEFVM1
			VM7900	IEFVMQMI
			VM7850	IEFVMPDQ
			VM7750	IEFVMVTE
			VM7742	
			VM7700	
			VM7370	
			VM7130	
			VM7090	
			VM7070	
			VM7060	
			VM7055AA	
			VM7055	
			VM7000	
			IEFVMQMI	
			IEFVMCVL	
			IEFVM	
		IEFVMLK5	IEFVMLK5	IEFVM6
		IEFVKMSG	IEFVKMJ1	IEFVKMSG
IEFVM1	IEFW21SD	IEFVKIMP	IEFVK	IEFVK
		CHLOADTB		CHLOADTB
		IEFACTFK	IEFACTLK	IEFACTLK
		IEFSD21Q	IEFW21SD	IEFW21SD
		IEFSD180	IEFSD18	IEFSD180
		IEFSD111	IEFSD111	IEFSD111
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFQMLK1	IEFQMSSS	IEFQMSSS
		IEFQASGQ	IEFQASGN	IEFQASNM
				IEFQASGN
		IEFCVFAK	IEFCVOL3	IEFCVOL1
			IEFCVOL2	
			IEFCVOL1	
		IEF41FAK	IEFW2FAK	IEFW41SD
			IEFW1FAK	
			IEFW41SD	
		IEFYSVMS	IEFYS	IEFYS
		IEFXCSSS	IEFXA	IEXAB00
				IEFXA
		IEFXAMSG	IEFXAMSG	IEFXAMSG
		IEFACTRT	IEFACTRT	IEFACTRT
		IEFWAD	IEFWAD	IEFWAD
				IEFSD572

(Continued)

Alias Name	Load Module Name	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
		IEFWAFAK		IEFWA000
		IEFSCAN	IEFSCAN	IEFSCAN
			UCBSCAN	
			UCBCONT	
		IEFDEVPT	IEFDEVPT	IEFDEVPT
		IEFWSTRT	IEFWSTRT	IEFWSTRT
		IEFWDFAK	IEFWD000	IEFWD000
		IEFWCFAK	IEFWC000	IEFWC000
		IEFVM76	VM7600	IEFVM76
		IEFVM5LS	VM7300	IEFVM5
		IEFVM4LS	VM7200	IEFVM4
		IEFVM3LS	VM7150	IEFVM3
		IEFVM2LS	VM7100	IEFVM2
		IEFVMLS1	VM7950	IEFVM1
			VM7900	
			VM7850	
			VM7750	
			VM7742	
			VM7700	
			VM7370	
			VM7130	
			VM7090	
			VM7070	
			VM7060	
			VM7055AA	
			VM7055	
			VM7000	
			IEFVMQMI	
			IEFVMCVL	
			IEFVM	
		IEFVMLK5	IEFVMLK5	IEFVM6
		IEFVKMSG	IEFVMJ1	IEFVKMSG
IEFVRRCA	IEFVRRC	IEFQASGQ	IEFQASGN	IEFQASGN
				IEFQASNM
		IEFSD514	IEFSD514	IEFSD514
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFVRRC	IEFVRRCB	IEFVRRC
			IEFVRRCA	
			IEFVRRC	
IEFVRRCB	IEFVRRC	IEFQASGQ	IEFQASGN	IEFQASGN
				IEFQASNM
		IEFSD514	IEFSD514	IEFSD514
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFVRRC	IEFVRRCB	IEFVRRC
			IEFVRRCA	
			IEFVRRC	
IEFVR2AE	IEFVRR2	IEFVRR2	IEFV2AE	IEFVRR2
			IEFVRR2	
IEFVR3AE	IEFVRR3	IEFVRR3	IEFV3AE	IEFVRR3
			IEFVRR3	
		IEFSD112	IEFSD112	IEFSD112
		IEFQDELQ	IEFQDELE	IEFQDELE
IEFV15XL	IEFVM6LS	IEFCVFAK	IEFCVOL1	IEFCVOL1
			IEFCVOL2	
			IEFCVOL3	
		IEFQMLK1	IEFQMSSS	IEFQMSSS
		IEFQASGQ	IEFQASGN	IEFQASNM
				IEFQASGN
		IEFVMLS6	IEFVMSGR	IEFVM6
		IEFSD41Q	IEFW2FAK	IEFW41SD
			IEFW1FAK	

(Continued)

Alias Name	Load Module Name	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
			IEFW41SD	
		IEFSD195	IEFVAWAT	IEFSD095
		IEFSD111	IEFSD111	IEFSD111
		IEFSD096	IEFSD096	IEFSD096
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFYSVMS	IEFYS	IEFYS
		IEFXKMSG	IEFXKMSG	IEFXKMSG
		IEFXKIMP	IEFXK000	IEFXK000
		IEFXJMSG	MSOFF	IEFXJMSG
			MSSYS	
			MSRCV	
		IEFXJIMP	IEFXJX5A	IEFXJ000
			IEFV15XL	
			IEFXJ000	
		IEFXAFAK	IEFXAFAK	IEFXA
		IEFVMMS1	IEFVM1	IEFVM1
		IEFVMLS7	IEFVM7	IEFVM7
IEFWC000	IEFWA000	IEFSD552	IEFSD552	IEFXJX5A
		IEFAVFAK		IEFXV001
		IEFWA000	IEFUCBL	IEFWA7
			IEFWA000	IEFWA002
		IEFX300A	X33B42	IEFX3000
			IEFX3000	
		IEFWSWIN	IEFWSWIT	IEFWSWIT
		IEFSGOPT	IEFSGOPT	IEFSGOPT
		IEFSCAN	IEFSCAN	IEFSCAN
			UCBSCAN	
			UCBCONT	
		IEFDEVPT	IEFDEVPT	IEFDEVPT
		IEFXDPTH	IEFXDPTH	IEFXDPTH
		IEFX5000	IEFX5000	IEFX5000
			X55D3G	
			X55C86	
			XIIB32	
		IEFWCIMP	IEFWC000	IEFWC000
		IEFXH000	IEFXH000	IEFXH000
		IEFWDFAK	IEFWD000	IEFWD000
		IEFXJFAK	IEFXJ000	IEFXJ000
		IEF41FAK	IEFW2FAK	IEFW41SD
			IEFW1FAK	
			IEFW41SD	
		IEFSD551	IEFSD551	IEFV15XL
IEFW41SD	IEFWD000	IEFWD000	IEFWD000	IEFWD000
				IEFWD002
				IEFWDMSG
		IEFWD001	IEFWD001	IEFWD001
		IEFWEXTA	IEFWEXTA	IEFWEXTA
		IEFXT00D	WDEXIT	IEFXT000
			XTTBC0	
			IEFXT000	
		IEFXTDMY	IEFXTDMY	IEFXTDMY
		IEFSD195	IEFVAWAT	IEFSD095
		IEFSD096	IEFSD096	IEFSD096
		IEFXT002	IEFXT002	IEFXT002
			XTTEA0	
			XTTEA1	
			XTTEB3	
			XTTRDJ	
		IEFSD097	IEFSD097	IEFSD097
		IEFQMLK1	IEFQMSSS	IEFQMSSS
		IEFYSVMS	IEFYS	IEFYS

(Continued)

Alias Name	Load Module Name	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFXT003	IEFXT003	IEFXT003
			XUUB00	
			XUUH06	
		IEFXKFAK	IEFXKFAK	IEFXK000
		IEFQASGQ	IEFQASGN	IEFQASGN
				IEFQASNM
		IEFSD41Q	IEFW2FAK	IEFW41SD
			IEFW1FAK	
			IEFW41SD	
		IEFXJFAK	IEFXJ000	IEFXJ000
		IEFVMMS1	IEFVM1	IEFVM1
		IEFSD111	IEFSD111	IEFSD111
				CHFILDRW
		IEFXTMSG	RESMSG	IEFXTMSG
			MSGDSDV	
			MESGHEAD	
			MESGDDEV	
			PRIAMSG	
			JOBCAN	
			WTORREP	
			RESUNIT	
		IEFCVFAK	IEFCVOL1	IEFCVOL1
			IEFCVOL2	
			IEFCVOL3	
IEFW42SD	IEFSD061	IEFDSTRT	IEFDSTRT	IEFDSTRT
		IEFDSTBL	IEFDSTBL	IEFDSTBL
		IEFACTRT	IEFACTRT	IEFACTRT
		IEFDSLST	IEFDSLST	IEFDSLST
		IEFQASGQ	IEFQASGN	IEFQASNM
				IEFQASGN
		IEFIDUMP	IEFIDUMP	IEFIDUMP
		IEFIDMPM	IEFIDMPM	IEFIDMPM
		IEFZGST1	ZGOJ8	IEFZG
			ZGOJ5	
			IEFZG	
		IEFYRCDS	IEFYRCDS	IEFSD572
				IEFYRCDS
		IEFYPMSG	YPPMSG2	IEFYPMSG
			YPPMSG1	
			IEFYPMSG	
		IEFYPJB3	IEFYP	IEFYP
		IEFYNMSG	STRMSG01	IEFYNMSG
			IEFYNMSG	
		IEFYNIMP	IEFYN	IEFYN
		IEFWTERM	IEFWTERM	IEFWTERM
		IEFVSDRA	IEFVSDRA	IEFVSDRA
		IEFVJMSG	IEFVJMSG	IEFVJMSG
		IEFVJIMP	IEFVJ	IEFVJ
		IEFUSI	IEFUSI	IEFUSI
		IEFUJI	IEFUJI	IEFUJI
		IEFSMFWI	IEFSMFWI	IEFSMFWI
		IEFSMFLK	IEFACTLK	IEFACTLK
		IEFSMFIE	IEFSMFIE	IEFSMFIE
		IEFSD514	IEFSD514	IEFSD514
		IEFSD42Q	IEFW42SD	IEFW42SD
		IEFSD31Q	IEFW31SD	IEFW31SD
		IEFSD22Q	IEFW22SD	IEFW22SD
		IEFSD168	IEFSD068	IEFSD068
		IEFSD166	IEFSD066	IEFSD066
		IEFSD164	IEFSD064	IEFSD064
		IEFSD161	IEFSD061	IEFDSOAL
				QMTMSD

(Continued)

Alias Name	Load Module Name	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
				IEFSD061
		IEFSD112	IEFSD112	IEFSD112
		IEFSD111	IEFSD111	IEFSD111
		IEFSD110	IEFSD110	IEFSD110
		IEFSD104	IEFSD104	IEFSD104
		IEFSD101	IEFSD101	IEFSD101
		IEFRPREP	IEFRPREP	IEFRPREP
		IEFQMUNQ	IEFQMUNC	IEFQMUNC
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFQMNQQ	IEFQMNQ2	IEFQMNQ2
		IEFQMLK1	IEFQMSSS	IEFQMSSS
		IEFQMDQQ	IEFQMDQ2	IEFQMDQ2
		IEFQDELQ	IEFQDELE	IEFQDELE
		IEF060SD	IEFSD060	IEFSD060
		IEFZHMSG	XPS631	IEFZH
			ZK0E1	
			ZK0D1	
			ZG0E60	
			IEFZH	
		IEFZGST2	ZPOQMGR1	IEFZG2
			ZPOC10	
			ZOOE10	
			ZOOA1	
			ZGOK09	
			IEFZG2	
		IEFSDPPT		IEFSDPPT
IEFXA	IEFW21SD	IEFVKIMP	IEFVK	IEFVK
		CHLOADTB		CHLOADTB
		IEFACTFK	IEFACTLK	IEFACTLK
		IEFSD21Q	IEFW21SD	IEFW21SD
		IEFSD180	IEFSD18	IEFSD180
		IEFSD111	IEFSD111	IEFSD111
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFQMLK1	IEFQMSSS	IEFQMSSS
		IEFQASGQ	IEFQASGN	IEFQASNM
				IEFQASGN
		IEFCVFAK	IEFCVOL3	IEFCVOL1
			IEFCVOL2	
			IEFCVOL1	
		IEF41FAK	IEFW2FAK	IEFW41SD
			IEFW1FAK	
			IEFW41SD	
		IEFYSVMS	IEFYS	IEFYS
		IEFXCSSS	IEFXA	IEXAB00
				IEFXA
		IEFXAMSG	IEFXAMSG	IEFXAMSG
		IEFACTRT	IEFACTRT	IEFACTRT
		IEFWAD	IEFWAD	IEFWAD
				IEFSD572
		IEFWAFAK		IEFWA000
		IEFSCAN	IEFSCAN	IEFSCAN
			UCBSCAN	
			UCBCONT	
		IEFDEVPT	IEFDEVPT	IEFDEVPT
		IEFWSTRT	IEFWSTRT	IEFWSTRT
		IEFWDFAK	IEFWD000	IEFWD000
		IEFWCFAK	IEFWC000	IEFWC000
		IEFVM76	VM7600	IEFVM76
		IEFVM5LS	VM7300	IEFVM5
		IEFVM4LS	VM7200	IEFVM4
		IEFVM3LS	VM7150	IEFVM3
		IEFVM2LS	VM7100	IEFVM2

(Continued)

Alias Name	Load Module Name	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
		IEFVMLS1	VM7950	IEFVM1
			VM7900	
			VM7850	
			VM7750	
			VM7742	
			VM7700	
			VM7370	
			VM7130	
			VM7090	
			VM7070	
			VM7060	
			VM7055AA	
			VM7055	
			VM7000	
			IEFVMQMI	
			IEFVMCVL	
			IEFVM	
		IEFVMLK5	IEFVMLK5	IEFVM6
		IEFVKMSG	IEFVKMJ1	IEFVKMSG
IEFXJX5A	IEFVM6LS	IEFCVFAK	IEFCVOL1	IEFCVOL1
			IEFCVOL2	
			IEFCVOL3	
		IEFQMLK1	IEFQMSSS	IEFQMSSS
		IEFQASGQ	IEFQASGN	IEFQASNM
				IEFQASGN
		IEFVMLS6	IEFVMSGR	IEFVM6
		IEFSD41Q	IEFW2FAK	IEFW41SD
			IEFW1FAK	
			IEFW41SD	
		IEFSD195	IEFVAWAT	IEFSD095
		IEFSD111	IEFSD111	IEFSD111
		IEFSD096	IEFSD096	IEFSD096
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFYSVMS	IEFYS	IEFYS
		IEFXKMSG	IEFXKMSG	IEFXKMSG
		IEFXKIMP	IEFXK000	IEFXK000
		IEFXJMSG	MSOFF	IEFXJMSG
			MSSYS	
			MSRCV	
		IEFXJIMP	IEFXJX5A	IEFXJ000
			IEFV15XL	
			IEFXJ000	
		IEFXAFAK	IEFXAFAK	IEFXA
		IEFVMMS1	IEFVM1	IEFVM1
		IEFVMLS7	IEFVM7	IEFVM7
IEFXJ000	IEFVM6LS	IEFCVFAK	IEFCVOL2	IEFCVOL1
			IEFCVOL3	
			IEFCVOL1	
		IEFQMLK1	IEFQMSSS	IEFQMSSS
		IEFQASGQ	IEFQASGN	IEFQASNM
				IEFQASGN
		IEFVMLS6	IEFVMSGR	IEFVM6
		IEFSD41Q	IEFW2FAK	IEFW41SD
			IEFW1FAK	
			IEFW41SD	
		IEFSD195	IEFVAWAT	IEFSD095
		IEFSD111	IEFSD111	IEFSD111
		IEFSD096	IEFSD096	IEFSD096
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFYSVMS	IEFYS	IEFYS
		IEFXKMSG	IEFXKMSG	IEFXKMSG
		IEFXKIMP	IEFXK000	IEFXK000

(Continued)

Alias Name	Load Module Name	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
		IEFXJMSG	MSOFF	IEFXJMSG
			MSSYS	
			MSRCV	
		IEFXJIMP	IEFXJX5A	IEFXJ000
			IEFV15XL	
			IEFXJ000	
		IEFXAFAK	IEFXAFAK	IEFXA
		IEFVMMS1	IEFVM1	IEFVM1
		IEFVMLS7	IEFVM7	IEFVM7
IEFXK000	IEFVM6LS	IEFCVFAK	IEFCVOL2	IEFCVOL1
			IEFCVOL3	
			IEFCVOL1	
		IEFQMLK1	IEFQMSSS	IEFQMSSS
		IEFQASGQ	IEFQASGN	IEFQASNM
				IEFQASGN
		IEFVMLS6	IEFVMSGR	IEFVM6
		IEFSD41Q	IEFW2FAK	IEFW41SD
			IEFW1FAK	
			IEFW41SD	
		IEFSD195	IEFVAWAT	IEFSD095
		IEFSD111	IEFSD111	IEFSD111
		IEFSD096	IEFSD096	IEFSD096
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFYSVMS	IEFYS	IEFYS
		IEFXKMSG	IEFXKMSG	IEFXKMSG
		IEFXKIMP	IEFXK000	IEFXK000
		IEFXJMSG	MSOFF	IEFXJMSG
			MSSYS	
			MSRCV	
		IEFXJIMP	IEFXJX5A	IEFXJ000
			IEFV15XL	
			IEFXJ000	
		IEFXAFAK	IEFXAFAK	IEFXA
		IEFVMMS1	IEFVM1	IEFVM1
		IEFVMLS7	IEFVM7	IEFVM7
IEFXV001	IEFXVAVR	IEFX5000	X55D3G	IEFX5000
			X55C86	
			XIIB32	
			IEFX5000	
		IEFQMLK1	IEFQMSSS	IEFQMSSS
		IEFCVFAK	IEFCVOL3	IEFCVOL1
			IEFCVOL2	
			IEFCVOL1	
		IEFX300A	X33B42	IEFX3000
			IEFX3000	
		IEFXV001	IEFXV001	IEFXV001
		IEFXV002	IEFXV002	IEFXV002
		IEFXV003	IEFXV003	IEFXV003
		IEFXVNSL	IEFXVNSL	IEFXVNSL
		IEFXVMSG	IEFXVMSG	IEFXVMSG
		IEFXJFAK	IEFXJ000	IEFXJ000
		IEFXH000	IEFXH000	IEFXH000
		IEFWD001	IEFWD001	IEFWD001
		IEFWD000	IEFWD000	IEFWD000
		IEFWCFAK	IEFWC000	IEFWC000
		IEFSD552	IEFSD552	IEFXJX5A
		IEFSD551	IEFSD551	IEFV15XL
		IEFSD111	IEFSD111	IEFSD111
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
IEF085SD	IEFSD085	IEFSD085	IEFSD085	IEFSD085
			IEF085SD	IEFSD85M
			IEF850SD	

(Continued)

Alias Name	Load Module Name	Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFSD171	IEFSD071	IEFSD071
				IEFSD71M
IEF086SD	IEFSD086	IEFSD089	IEFSD089	IEFSD89M
				IEFSD089
		IEFSD088	IEFSD088	IEFSD088
		IEFSD086	IEF086SD	IEFSD86M
			IEFSD086	IEFSD086
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
IEF850SD	IEFSD085	IEFSD171	IEFSD071	IEFSD71M
				IEFSD071
		IEFQMRAW	IEFQMRAW	IEFQMRAW
				IEFQMWTO
		IEFSD085	IEFSD085	IEFSD085
			IEF850SD	IEFSD85M
LOC	IEFLOCDQ	IEFCNVRT	CNVRT	IEFCNVRT
		IEFRDWRT	WRT	IEFRDWRT
			RD	
		IEFLOCDQ	LOC	IEFLOCDQ
			LOCCAN	
			LOCDQ	
LOCCAN	IEFLOCDQ	IEFCNVRT	CNVRT	IEFCNVRT
		IEFRDWRT	WRT	IEFRDWRT
			RD	
		IEFLOCDQ	LOC	IEFLOCDQ
			LOCCAN	
			LOCDQ	
LOCDQ	IEFLOCDQ	IEFCNVRT	CNVRT	IEFCNVRT
		IEFRDWRT	WRT	IEFRDWRT
			RD	
		IEFLOCDQ	LOC	IEFLOCDQ
			LOCCAN	
			LOCDQ	

Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name	Load Module Name	Alias Name
CHLOADTB		CHLOADTB	IEFW21SD	IEFXA IEFVM1 IEFVMCVL
IEECVINT	IEECVCTI	IEECVCTI	IEECVCTI	
IEELWAIT	IEELWAIT	IEELWAIT	IEELWAIT	
IEEPALTR	IEEPALTR	IEEPSTP1	IEEPALTR	
IEEPDISC	IEEPDISC	IEEPDISC	IEEPDISC	
IEEPPRES	IEEPPRES	IEEPPRES	IEEPPRES	
IEEPRTN2	IEEPRTN2	IEEPRTN2	IEEPRTN	
IEEPRWI2	IEEPRWI2	IEEPRWI2	IEEPRWI2	
IEEPSN	IEEPSN	IEEPSN	IEEVRJCL	IEEPSN
IEESD562	IEESD562	IEESD562	IEEVDNX1	IEESD562
IEESD563	IEESD563	IEESD563	IEESD563	
IEESD564	IEESD564	IEESD564	IEESD564	
IEESD565	IEESD565	IEESD565	IEESD565	
IEESD575	IEESD575	IEESD575	IEESD575	IEESD581
IEESD576	IEESD576	IEESD576	IEESD576	IEESD580
IEESD577	IEESD577	IEESD577	IEESD577	
IEESD578	IEESD578	IEESD578	IEESD578	
IEESD579	IEESD579	IEEMSWTO IEESD579	IEESD579	
IEESD580	IEESD580	IEESD580	IEESD576	IEESD580
IEESD581	IEESD581	IEESD581	IEESD575	IEESD581
IEESD582	IEESD582 IEESD82A	IEESD582	IEESD582	IEESD82A
IEESD583	IEESD583	IEESD583	IEESD583	
IEESD584	IEESD584	IEESD584	IEESD584	
IEESMFAL	IEESMFAL IEESMFI4	IEESMFAL	IGC0208C	
IEESMFI2	IEESMFI2	IEESMFI2	IEESMFI2	
IEESMFI3	IEESMFI0 IEESMFI3 IEESMFMS	IEESMFI3	IEESMFI3	IEESMFMS IEESMFI0
IEESMFOI	IEESMFOI	IEESMFOI	IEESMFOI	
IEESMFOP	IEESMFOP	IEESMFOP	IGC0108C	
IEESMFWR	IEESMFWR	IEESMFWR	IEESMFWR	
IEESMF8C	IEESMF8C	IGC0083	IGC0008C	
IEEVDRGN	IEEVDRGN	IEEVDRGN	IEEVDSP1	
IEEVDSP1	IEEVDSP1	IEEVDSP1	IEEVDSP1	
IEEVICLR	IEEVICLR IEFICR	IEEVICLR	IEEVICLR	IEFICR
IEEVICTL	IEEVICTL	IEEVICTL	IEFVICTL	IEEVIC
IEEVIPL	IEEVIPL	IEEVIPL	IEEVIPL	
IEEVJCL	IEEVJCL	IEEVJCL	IEEVMNT1 IEEVSTAR	
IEEVLDSP	IEEVLDSP	IEEVLDSP	IEEVLDSP	
IEEVLIN1	IEEVLIN1	IEEVLIN1	IEEVLIN	
IEEVLNKT		IEEVLNKT	IEEVRJCL	IEEPSN
IEEVLOUT	IEEVLOUT	IEEVLOUT	IEEVLOUT	
IEEVMNT1	IEEVMNT1	IEEVMNT1	IEEVMNT1	
IEEVMNT2	IEEVMNT2	IEEVMNT2	IEEVMNT2	
IEEVOMSG	IEEVOMSG	IEEVOMSG	IEEVOMSG	
IEEVPRES	IEEVPRES	IEEVPRES	IEEVPRES	
IEEVRC	IEEVRC	IEEVRC	IEEVRC	
IEEVRCTL	IEEVRCTL	IEEVRCTL	IEEVRCTL	
IEEVRFRX	IEEVRFRX	IEEVRFRX	IEEVLIN IEECVCTL	
IEEVRJCL	IEEVRJCL	IEEVRJCL	IEEVRJCL	IEEPSN
IEEVSIPL	IEEVSIPL	IEEVSIPL	IEEVSIPL	
IEEVSMSG	IEEVSMSG	IEEVSMSG	IEEVICTL IEEVMNT2 IEFIRC	
IEEVSTAR	IEEVSTAR	IEEVJCL IEEVSTAR	IEEVSTAR	
IEEVWAIT	IEEVWAIT	IEEVWAIT	IEEVWAIT	IEEVWDAR
IEEXEDNA	IEEXEDNA	IEEXEDNA	IEEXEDNA	
IEE0003D	IEE0003D	IEE0003D	IGC0003D	
IEE00110	IEE00110	IEE00110	IGC00110	
IEE0303D	IEE0303D	IEE0303D	IGC0303D	
IEE0403D	IEE0403D	IEE0403D	IGC0403D	
IEE0503D	IEE0503D	IEE0503D	IGC0503D IEE0503D	
IEE0603D	IEE0603D	IEE0603D	IGC0603D	
IEE0703D	IEE0703D	IEE0703D	IGC0703D	
IEE0803D	IEE0803D	IEE0803D	IGC0803D	
IEE0903D	IEAQOT00	IEAQOT00	IGC0903D	
IEE1A03D	IEE1A03D	IEE1A03D	IGC1203D	
IEE1B03D	IEE1B03D	IEE1B03D	IGC1B03D	
IEE10110	IEE10110	IEE10110	IGC10110	
IEE1103D	IEE1103D	IEE1103D	IGC1103D	
IEE11110	IEE11110	IEE11110	IGC11110	
IEE1203D	IEE1203D	IEE1203D	IGC1203D	
IEE12110	IEE12110	IEE12110	IGC12110	
IEE1403D	IEE1403D	IEE1403D	IGC1403D	
IEE1603D	IEE1603D	IEE1603D	IGC1603D	
IEE20110	IEE20110	IEE20110	IGC20110	
IEE2103D	IEE2103D	IEE2103D	IGG2103D	
IEE21110	IEE21110	IEE21110	IGC21110	
IEE22110	IEE22110	IEE22110	IGC22110	
IEE2303D	IEE2303D	IEE2303D	IGC2303D	
IEE23110	IEE23110	IEE23110	IGC23110	
IEE2903D	IEE2903D	IEE2903D	IGC2903D	
IEE30110	IEE30110	IEE30110	IGC30110	
IEE3103D	IEE3103D	IEE3103D	IGC3103D	
IEE31110	IEE31110	IEE31110	IGC31110	
IEE3203D	IEE3203D	IEE3203D	IGC3203D	
IEE32110	IEE32110	IEE32110	IGC32110	
IEE3303D	IEE3303D	IEE3303D	IGC3303D	
IEE3503D	IEE3503D	IEE3503D	IGC3503D	
IEE3603D	IEE3603D	IEE3603D	IGC3603D	
IEE3703D	IEE3703D	IEE3703D	IGC3703D	
IEE3803D	IEE3803D	IEE3803D	IGC3803D	
IEE40110	IEE40110	IEE40110	IGC40110	
IEE4103D	IEE4103D	IEE4103D	IGC4103D	
IEE4203D	IEE4203D	IEE4203D	IGC4203D	
IEE4303D	IEE4303D	IEE4303D	IGC4303D	

(Continued)

Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name	Load Module Name	Alias Name
IEE4403D	IEE4403D	IEE4403D	IGC4403D	
IEE4503D	IEE4503D	IEE4503D	IGC4503D	
IEE4603D	IEE4603D	IEE4603D	IGC4603D	
IEE4703D	IEE4703D	IEE4703D	IGC4703D	
IEE4803D	IEE4803D	IEE4803D	IGC4803D	
IEE4903D	IEE4903D	IEE4903D	IGC4903D	
IEE50110	IEE50110	IEE50110	IGC50110	
IEE5103D	IEE5103D	IEE5103D	IGC5103D	
IEE5203D	IEE5203D	IEE5203D	IGC5203D	
IEE5303D	IEE5303D	IEE5303D	IGC5303D	
IEE5403D	IEE5403D	IEE5403D	IGC5403D	
IEE5503D	IEE5503D	IEE5503D	IGC5503D	
IEE5603D	IEE5603D	IEE5603D	IGC5603D	
IEE5903D	IEE5903D	IEE5903D	IGC5903D	
IEE60110	IEE60110	IEE60110	IGC60110	
IEE6303D	IEE6303D	IEE6303D	IGC6303D	
IEE6403D	IEE6403D	IEE6403D	IGC6403D	
IEE6503D	IEE6503D	IEE6503D	IGC6503D	
IEE6603D	IEE6603D	IEE6603D	IGC6603D	
IEE6703D	IEE6703D	IEE6703D	IGC6703D	
IEE6803D	IEE6803D	IEE6803D	IGC6803D	
IEE6903D	IEE6903D	IEE6903D	IGC6903D	
IEE7103D	IEE7103D	IEE7103D	IGC7103D	
IEE7203D	IEE7203D	IEE7203D	IGC7203D	
IEE7303D	IEE7303D	IEE7303D	IGC7303D	
IEE7503D	IEE7503D	IEE7503D	IGC7503D	
IEE7603D	IEE7603D	IEE7603D	IGC7603D	
IEE7703D	IEE7703D	IEE7703D	IGC7703D	
IEE7803D	IEE7803D	IEE7803D	IGC7803D	
IEE7903D	IEE7903D	IEE7903D	IGC7903D	
IEE8503D	IEE8503D	IEE8503D	IGC8503D	
IEE8603D	IEE8603D	IEE8603D	IGC8603D	
IEFACT	IEFACT	IEFACT	IEFVHA	IEFVHCB
IEFACTFK	IEFACTLK	IEFACTLK	IEFW21SD	IEFVM1
				IEFXA
				IEFVMCVL
IEFACTRT	IEFACTRT	IEFACTRT	IEFSD061	IEFSD064
				IEFSD104
				IEFW42SD
			IEFW21SD	IEFVMCVL
				IEFVM1
				IEFXA
IEFAVFAK		IEFXV001	IEFWA000	IEFWC000
IEFBR14	IEFBR14	IEFBR14	IEFBR14	
IEFCNVRT	CNVRT	IEFCNVRT	IEFLOCDQ	LOC
				LOCCAN
				LOCDQ
			IEFVRR1	
IEFCVFAK	IEFCVOL1	IEFCVOL1	IEFW21SD	IEFVMCVL
	IEFCVOL3			IEFXA
	IEFCVOL2			IEFVM1
			IEFVM6LS	IEFXJ000
				IEFXJX5A
				IEFV15XL
				IEFXK000
			IEFWD000	IEFW41SD
			IEFXVAVR	IEFXV001
IEFDEVPT	IEFDEVPT	IEFDEVPT	IEFWA000	IEFWC000
			IEFW21SD	IEFVMCVL
				IEFVM1
				IEFXA
IEFDSDRP	IEFDSDRP	IEFDSDRP	IEFDSDRP	
IEFDSLST	IEFDSLST	IEFDSLST	IEFSD061	IEFSD104
				IEFSD064

(Continued)

Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name	Load Module Name	Alias Name
				IEFW42SD
IEFDSOAL	IEFDSOAL	IEFDSOAL	IEFDSOAL	
IEFDSOCP	IEFDSOCP	IEFDSOCP	IEFDSO	
IEFDSOCR	IEFDSOCR	IEFDSOCR	IEFDSOCR	
IEFDSOFB	IEFDSOFB	IEFDSOFB	IEFDSOFB	
IEFDSOLP	IEFDSOLP	IEFDSOLP	IEFDSOLP	
IEFDSOSM	IEFDSOSM	IEFDSOSM	IEFDSOSM	
IEFDSOWR	IEFDSOWR	IEFDSOWR	IEFDSOWR	
IEFDSTBL	IEFDSTBL	IEFDSTBL	IEFSD061	
				IEFSD064
				IEFSD104
				IEFW42SD
IEFDSTRT	IEFDSTRT	IEFDSTRT	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFIDMPM	IEFIDMPM	IEFIDMPM	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFIDUMP	IEFIDUMP	IEFIDUMP	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFIIC	IEFIIC	IEFIIC	IEFIIC	
IEFIRC	IEFIRC	IEFIRC	IEFIRC	
IEFIRCB	IEFIRCB	IEFIRCB	IEFQMSSS	IEFIRCB
IEFK1MSG	IEFK1MSG	IEFK1MSG	IEEVPRES	
IEFLOCDQ	LOCDQ	IEFLOCDQ	IEFLOCDQ	LOCDQ
	LOC			LOCCAN
	LOCCAN			LOC
			IEFVRR1	
IEFMCVOL	IEFCVOL3	IEFCVOL3	IEFMCVOL	IEFCVOL3
	IEFCVOL2	IEFCVOL2		IEFCVOL2
	IEFCVOL1	IEFCVOL1		IEFCVOL1
IEFORMAT	IEFORMAT	IEFORMAT	IEFSQINT	
IEFPRES	IEFPRES	IEFPRES	IEEVPRES	
IEFPRTXX	SPRINTER	SPRINTER	IEFPRINT	
IEFQAGST	IEFQAGST	IEFQAGST	IEFQMSSS	IEFIRCB
			IEFVMF	
			IEFVMB	
IEFQASGQ	IEFQASGN	IEFQASNM	IEFVMB	
		IEFQASGN		
			IEFVMF	
			IEFSD304	
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFSD062	IEFV4221
			IEFQMSSS	IEFIRCB
			IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
			IEFDSDRP	
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
			IEFVRRC	IEFVRRCB
				IEFVRRCA
			IEFWD000	IEFW41SD
IEFQBVMS	IEFQMSSS	IEFQMSSS	IEFQMSSS	IEFIRCB
			IEFVMF	
			IEFDSDRP	

(Continued)

Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name	Load Module Name	Alias Name
IEFQDELQ	IEFQDELE	IEFQDELE	IEFQDELE	
			IEFVMF	
			IEFVMD	
			IEFSD080	IEFSD079
				IEEVWTR1
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFSD304	
			IEFVRR3	IEFVR3AE
			IEESD576	IEESD580
IEFQINTZ	IEFQINTZ	IEFQINTZ	IEFQINTZ	
IEFQMDQQ	IEFQMDQ2	IEFQMDQ2	IEFQMDQQ	IEFQMDQ2
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFSD080	IEFSD079
				IEEVWTR1
IEFQMDUM		IEFQMSSS	IEFVHA	IEFVHCB
IEFQMLK1	IEFQMSSS	IEFQMSSS	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
			IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFWD000	IEFW41SD
			IEFXVAVR	IEFXV001
IEFQMNQQ	IEFQMNQ2	IEFQMNQ2	IEFQMSSS	IEFIRCB
			IEFQMNQ2	
			IEFDSOCR	
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFSD304	
			IEFVMF	
			IEFVMB	
			IEESD577	
			IEESD579	
IEFQMRAW	IEFQMRAW	IEFQMRAW	IEFVMF	
		IEFGMWTO	IEFSD304	
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFSD062	IEFV4221
			IEFSD086	IEF086SD
			IEFSD085	IEFSD071
				IEF085SD
				IEF850SD
			IEFDSOCR	
			IEFDSO	
			IEFDSDRP	
			IEFQMRAW	
			IEFQMSSS	IEFIRCB
			IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
			IEFXVAVR	IEFXV001

(Continued)

Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name	Load Module Name	Alias Name
			IEFWD000	IEFW41SD
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
			IEFVRRC	IEFVRRCB
				IEFVRRCA
IEFQMSGV		IEFQMSGV	IEFSQINT	
IEFQMUNQ	IEFQMUNC	IEFQMUNC	IEFSD080	IEFSD079
				IEEVWTR1
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFVMF	
			IEFQMUNC	
IEFRCLN1	IEFRCLN1	IEFRCLN1	IEFRCLN1	
IEFRCLN2	IEFRCLN2	IEFRCLN2	IEFRCLN2	
IEFRDWRT	WRT	IEFRDWRT	IEFLOCDQ	LOCCAN
	RD			LOC
				LOCDQ
			IEFVRR1	
IEFRPREP	IEFRPREP	IEFRPREP	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFRSTRT	IEFSMR	IEFRSTRT	IEFRSTRT	IEFSMR
	IEFRSTRT			
IEFSCAN	IEFSCAN	IEFSCAN	IEFW21SD	IEFVMCVL
	UCBSCAN			IEFVM1
	UCBCONT			IEFXA
			IEFWA000	IEFWC000
IEFSDPPT		IEFSDPPT	IEFSD061	IEFSD065
				IEFW42SD
				IEFSD104
IEFSDTTE	IEFSDTTE	IEFSDTTE	IEFSDTTE	
IEFSDXXX	IEFSDXXX	IEFSDXXX	IEFSDXXX	
IEFSDXYZ	IEFSDXYZ	IEFSDXYZ	IEFSDXYZ	
IEFSD0XX	IEFSD0XX	IEFSD0XX	IEFSD0XX	
IEFSD055	IEFSD055	IEFSD055	IEFSQINT	
IEFSD070	IEFSD070	IEFSD070	IEFSD070	
IEFSD078	IEFSD078	IEFSD078	IEFSD078	
IEFSD079	IEFSD079	IEFSD079	IEFSD080	IEFSD079
				IEEVWTR1
IEFSD080	IEFSD080	IEFSD080	IEFSD080	IEFSD079
				IEEVWTR1
IEFSD081	IEFSD081	IEFSD081	IEFSD080	IEFSD079
				IEEVWTR1
IEFSD082	IEFSD082	IEFSD082	IEFSD080	IEFSD079
				IEEVWTR1
IEFSD083	IEFSD083	IEFSD83M	IEFSD080	IEFSD079
		IEFSD083		IEEVWTR1
IEFSD084	IEFSD084	IEFSD084	IEFSD080	IEFSD079
				IEEVWTR1
IEFSD085	IEF850SD	IEFSD85M	IEFSD085	IEF850SD
	IEF085SD	IEFSD085		IEFSD071
	IEFSD085			IEF085SD
IEFSD086	IEFSD086	IEFSD086	IEFSD086	IEF086SD
	IEF086SD	IEFSD86M		
IEFSD087	IEFSD087	IEFSD87M	IEFSD087	
		IEFSD087		
IEFSD088	IEFSD088	IEFSD088	IEFSD087	
			IEFSD094	

(Continued)

Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name	Load Module Name	Alias Name
			IEFSD086	IEF086SD
IEFSD089	IEFSD089	IEFSD89M	IEFSD086	IEF086SD
		IEFSD089		
			IEFSD094	
			IEFSD087	
IEFSD094	IEFSD094	IEFSD094	IEFSD094	
IEFSD095	IEFSD095	IEFSD095	IEFSD094	
IEFSD096	IEFSD096	IEFSD096	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFWD000	IEFW41SD
IEFSD097	IEFSD097	IEFSD097	IEFWD000	IEFW41SD
IEFSD101	IEFSD101	IEFSD101	IEFSD061	IEFW42SD
				IEFSD104
				IEFSD064
IEFSD102	IEFSD102	IEFSD102	IEFSD102	
IEFSD103	IEFSD063	IEFSD063	IEFSD062	IEFV4221
IEFSD104	IEFSD104	IEFSD104	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFSD105	IEFSD105	IEFSD105	IEFSD105	
IEFSD110	IEFSD110	IEFSD110	IEFSD062	IEFV4221
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFVMF	
IEFSD111	IEFSD111	IEFSD111	IEFVMF	
		CHF1LDRW		
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFSD062	IEFV4221
			IEFWD000	IEFW41SD
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
			IEFXVAVR	IEFXV001
			IEFDSOCR	
			IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
IEFSD112	IEFSD112	IEFSD112	IEFDSOCR	
			IEFVRR3	IEFVR3AE
			IEFSD062	IEFV4221
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFVMF	
IEFSD160	IEFSD060	IEFSD60M	IEFSD060	
		IEFSD060		
IEFSD161	IEFSD061	IEFDSOAL	IEFSD061	IEFW42SD
		QMTMSD		IEFSD064
		IEFSD061		IEFSD104
IEFSD162	IEFSD062	IEFSD062	IEFSD062	IEFV4221
IEFSD164	IEFSD064	IEFSD064	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFSD166	IEFSD066	IEFSD066	IEFSD061	IEFW42SD
				IEFSD064

(Continued)

Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name	Load Module Name	Alias Name
				IEFSD104
IEFSD168	IEFSD068	IEFSD068	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFDSOCR	
IEFSD171	IEFSD071	IEFSD71M	IEFSD085	IEF850SD
		IEFSD071		IEF085SD
				IEFSD071
IEFSD180	IEFSD18	IEFSD180	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFSD195	IEFVAWAT	IEFSD095	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFWD000	IEFW41SD
IEFSD21Q	IEFW21SD	IEFW21SD	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFSD22Q	IEFW22SD	IEFW22SD	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFSD263	IEFSD263	IEFSD263	IEFSD263	
IEFSD300	IEFSD300	IEFSD300	IEFSD300	
IEFSD301	IEFSD301	IEFSD301	IEFSD300	
IEFSD302	IEFSD302	IEFSD302	IEFSD300	
IEFSD303	IEFSD303	IEFSD303	IEFSD300	
IEFSD304	IEFSD304	IEFSD304	IEFSD304	
IEFSD305	IEFSD305	IEFSD305	IEFSD304	
IEFSD308	IEFSD308	IEFSD308	IEFSD308	
IEFSD31Q	IEFW31SD	IEFW31SD	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFSD310	IEFSD310	IEFSD310	IEFSD304	
			IEFSD300	
IEFSD311	SD55MSG3	IEFSD311	IEFSQINT	
	SD55MSG2			
	SD55MSG1			
	IEFSD311			
IEFSD312	SD304MG2	IEFSD312	IEFSD304	
	SD304MG1			
	IEFSD312			
	SD305MG1			
			IEFSD300	
IEFSD41Q	IEFW2FAK	IEFW41SD	IEFVM6LS	IEFXK000
	IEFW1FAK			IEFXJ000
	IEFW41SD			IEFXJX5A
				IEFV15XL
			IEFWD000	IEFW41SD
IEFSD42Q	IEFW42SD	IEFW42SD	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFSD514	IEFSD514	IEFSD514	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFSD304	
			IEFSD062	IEFV4221
			IEFVRRC	IEFVRRCB
				IEFVRRCA
IEFSD551	IEFSD551	IEFV15XL	IEFWA000	IEFWC000
			IEFXVAVR	IEFXV001
IEFSD552	IEFSD552	IEFXJX5A	IEFXVAVR	IEFXV001
			IEFWA000	IEFWC000
IEFSGOPT	IEFSGOPT	IEFSGOPT	IEFWA000	IEFWC000

(Continued)

Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name	Load Module Name	Alias Name
IEFSMFAT	IEFSMFAT	IEFSMFAT	IEFSD263	
IEFSMFIE	IEFSMFIE	IEFSMFIE	IEFSD061	IEFW42SD IEFSD064 IEFSD104
IEFSMFLK	IEFACTLK	IEFACTLK	IEFSD061	IEFW42SD IEFSD064 IEFSD104
IEFSMFWI	IEFSMFWI	IEFSMFWI	IEFSD061	IEFW42SD IEFSD064 IEFSD104
IEFUJI	IEFUJI	IEFUJI	IEFSD061	IEFW42SD IEFSD064 IEFSD104
IEFUJV	IEFUJV	IEFUJV	IEFUJV	
IEFUSI	IEFUSI	IEFUSI	IEFSD061	IEFW42SD IEFSD064 IEFSD104
IEFUSO	IEFUSO	IEFUSO	IEFSD263	
IEFUTL	IEFUTL	IEFUTL	IEFSD263	
IEFVDA	IEFVDA	IEFVDA	IEFVHA	IEFVHCB
IEFVDBSD	IEFVDBSD	IEFVDBSD	IEFVHA	IEFVHCB
IEFVEA	IEFVEA	IEFVEA	IEFVHA	IEFVHCB
IEFVFA	IEFVFA	IEFVFA	IEFVHA	IEFVHCB
IEFVFB	IEFVFB	IEFVFB	IEFVHA	IEFVHCB
IEFVGI	IEFVGI	IEFVGI	IEFVHA	IEFVHCB
IEFVGK	IEFVGK	IEFVGK	IEFVHA	IEFVHCB
IEFVGM	IEFVGM	IEFVGM	IEFVHA IEFVINA	IEFVHCB
IEFVGM1		IEFVGM1	IEFVGM1	
IEFVGM10		IEFVGM10	IEFVGM10	
IEFVGM11		IEFVGM11	IEFVGM11	
IEFVGM12		IEFVGM12	IEFVGM12	
IEFVGM13		IEFVGM13	IEFVGM13	
IEFVGM14		IEFVGM14	IEFVGM14	
IEFVGM15		IEFVGM15	IEFVGM15	
IEFVGM16		IEFVGM16	IEFVGM16	
IEFVGM17		IEFVGM17	IEFVGM17	
IEFVGM18		IEFVGM18	IEFVGM18	
IEFVGM19		IEFVGM19	IEFVGM19	
IEFVGM2		IEFVGM2	IEFVGM2	
IEFVGM3		IEFVGM3	IEFVGM3	
IEFVGM4		IEFVGM4	IEFVGM4	
IEFVGM5		IEFVGM5	IEFVGM5	
IEFVGM6		IEFVGM6	IEFVGM6	
IEFVGM7		IEFVGM7	IEFVGM7	
IEFVGM70		IEFVGM70	IEFVGM70	
IEFVGM71		IEFVGM71	IEFVGM71	
IEFVGM78		IEFVGM78	IEFVGM78	
IEFVGM8		IEFVGM8	IEFVGM8	
IEFVGM9		IEFVGM9	IEFVGM9	
IEFVGS	IEFVGS	IEFVGS	IEFVHA	IEFVHCB
IEFVGT	IEFVGT	IEFVGT	IEFVHA	IEFVHCB
IEFVHA	IEFVHA	IEFVHA	IEFVHA	IEFVHCB
IEFVHAA	IEFVHAA	IEFVHAA	IEFVHA	IEFVHCB
IEFVHB	IEFVHB	IEFVHB	IEFVHA	IEFVHCB
IEFVHC	IEFVHC	IEFVHC	IEFVHA	IEFVHCB
IEFVHCB	IEFVHCB	IEFVHCB	IEFVHA	IEFVHCB
IEFVHE	IEFVHE	IEFVHE	IEFVHA	IEFVHCB
IEFVHEB	IEFVHEB	IEFVHEB	IEFVHA	IEFVHCB
IEFVHEC	IEFVHEC	IEFVHEC	IEFVHA	IEFVHCB
IEFVHF	IEFVHF	IEFVHF	IEFVHA	IEFVHCB
IEFVHG	IEFVHG	IEFVHG	IEFVHA	IEFVHCB
IEFVHH	IEFVHH	IEFVHH	IEFVHA	IEFVHCB
IEFVHHB	IEFVHHB	IEFVHHB	IEFVHA	IEFVHCB

(Continued)

Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name	Load Module Name	Alias Name
IEFVHL	IEFVHL	IEFVHL	IEFVHA	IEFVHCB
IEFVHM	IEFVHM	IEFVHM	IEFVHA	IEFVHCB
IEFVHN	IEFVHN	IEFVHN	IEFVHN	
IEFVHQ	IEFVHQ	IEFVHQ	IEFVHA IEFVINA	IEFVHCB
IEFVHR	IEFVHR	IEFVHR	IEFVHA IEFVINA	IEFVHCB
IEFVH1	IEFVH1	IEFVH1	IEFVH1	
IEFVH2	IEFVH2	IEFVH2	IEFVH1	
IEFVINA	IEFVINA	IEFVINA	IEFVINA	
IEFVINB	IEFVINB	IEFVINB	IEFVINA IEFVINB	
IEFVINC	IEFVINC	IEFVINC	IEFVINA	
IEFVIND	IEFVIND	IEFVIND	IEFVHA	IEFVHCB
IEFVINE	IEFVINE	IEFVINE	IEFVINA	
IEFVJA	IEFVJA	IEFVJA	IEFVHA	IEFVHCB
IEFVJIMP	IEFVJ	IEFVJ	IEFSD061	IEFW42SD IEFSD064 IEFSD104
IEFVJMSG	IEFVJMSG	IEFVJMSG	IEFSD061	IEFW42SD IEFSD064 IEFSD104
IEFVKG	IEFVKG	IEFVKG	IEFVHA	IEFVHCB
IEFVKIMP	IEFVK	IEFVK	IEFW21SD	IEFVMCVL IEFXA IEFVM1
IEFVKMSG	IEFVKMJ1	IEFVKMSG	IEFW21SD	IEFVMCVL IEFXA IEFVM1
IEFVMA	IEFVMA	IEFVMA	IEFVMA	
IEFVMB	IEFVMB	IEFVMB	IEFVMB	
IEFVMC	IEFVMC	IEFVMC	IEFVMC	
IEFVMD	IEFVMD	IEFVMD	IEFVMD	
IEFVME	IEFVME	IEFVME	IEFVMF	
IEFVMF	IEFVMF	IEFVMF	IEFVMF	
IEFVMFAK	IEFVMCVL	IEFVMCVL	IEFMCVOL	IEFCVOL3 IEFCVOL2 IEFCVOL1
IEFVMG	IEFVMG	IEFVMG	IEFVMF	
IEFVMH	IEFVMH	IEFVMH	IEFVMF	
IEFVMLK5	IEFVMLK5	IEFVM6	IEFW21SD	IEFVMCVL IEFXA IEFVM1
IEFVMLS1	IEFVMCVL IEFVM VM7750 VM7742 VM7700 VM7370 VM7130 VM7090 VM7070 VM7060 VM7055AA VM7055 VM7000 IEFVMQMI VM7950 VM7900 VM7850	IEFVM1 IEFVMVTE IEFVMQMI IEFVMPDQ	IEFW21SD	IEFVMCVL IEFXA IEFVM1
IEFVMLS6	IEFVMSGR	IEFVM6	IEFVM6LS	IEFXK000 IEFXJ000 IEFXJX5A IEFV15XL

(Continued)

Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name	Load Module Name	Alias Name
			IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
IEFVMLS7	IEFVM7	IEFVM7	IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
IEFVMMS1	IEFVM1	IEFVM1	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFWD000	IEFW41SD
			IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
IEFVM2LS	VM7100	IEFVM2	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFVM3LS	VM7150	IEFVM3	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFVM4LS	VM7200	IEFVM4	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFVM5LS	VM7300	IEFVM5	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFVM76	VM7600	IEFVM76	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFVRRC	IEFVRRCB	IEFVRRC	IEFVRRC	IEFVRRCB
	IEFVRRCA			IEFVRRCA
	IEFVRRC			
IEFVRR1	IEFVRR1	IEFVRR1	IEFVRR1	
IEFVRR2	IEFV2AE	IEFVRR2	IEFVRR2	IEFVR2AE
	IEFVRR2			
IEFVRR3	IEFV3AE	IEFVRR3	IEFVRR3	IEFVR3AE
	IEFVRR3			
IEFVSDRA	IEFVSDRA	IEFVSDRA	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFVSDRD	IEFVSDRD	IEFVSDRD	IEFSD304	
IEFVSD12	IEFSD012	IEFSD012	IEFVHA	IEFVHCB
IEFVSD13	IEFSD090	IEFSD090	IEFVHA	IEFVHCB
IEFVSMBR	IGC0005B	IGC0005B	IGC0005B	
IEFVSMSG	IEFVSMSG	IEFVSMSG	IEFVH1	
IEFWAD	IEFWAD	IEFWAD	IEFW21SD	IEFVMCVL
		IEFSD572		IEFVM1
				IEFXA
IEFWAFAK		IEFWA000	IEFW21SD	IEFVMCVL
				IEFVM1
				IEFXA
IEFWA000	IEFUCBL	IEFWA7	IEFWA000	IEFWC000
	IEFWA000	IEFWA002		
IEFWCFAK	IEFWC000	IEFWC000	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
			IEFXVAVR	IEFXV001
IEFWCIMP	IEFWC000	IEFWC000	IEFWA000	IEFWC000
IEFWDFAK	IEFWD000	IEFWD000	IEFWA000	IEFWC000
IEFWD000	IEFWD000	IEFWD000	IEFXVAVR	IEFXV001

(Continued)

Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name	Load Module Name	Alias Name
			IEFWD000	IEFW41SD
IEFWD001	IEFWD001	IEFWD001	IEFWD000	IEFW41SD
			IEFXVAVR	IEFXV001
IEFWEXTA	IEFWEXTA	IEFWEXTA	IEFWD000	IEFW41SD
IEFWSTRT	IEFWSTRT	IEFWSTRT	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFWSWIN	IEFWSWIT	IEFWSWIT	IEFWA000	IEFWC000
IEFWTERM	IEFWTERM	IEFWTERM	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFWTP00	IGC0203E	IGC0203E	IGC0203E	
IEFWTP01	IGC0303E	IGC0303E	IGC0303E	
IEFWTP02	IGC0403E	IGC0403E	IGC0403E	
IEFXAFAK	IEFXAFAK	IEFXA	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
IEFXAMSG	IEFXAMSG	IEFXAMSG	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFXCSSS	IEFXA	IEXAB00	IEFW21SD	IEFVMCVL
		IEFXA		IEFXA
				IEFVM1
IEFXDPTH	IEFXDPTH	IEFXDPTH	IEFWA000	IEFWC000
IEFXH000	IEFXH000	IEFXH000	IEFWA000	IEFWC000
			IEFXVAVR	IEFXV001
IEFXJFAK	IEFXJ000	IEFXJ000	IEFXVAVR	IEFXV001
			IEFWA000	IEFWC000
			IEFWD000	IEFW41SD
IEFXJIMP	IEFXJ000	IEFXJ000	IEFVM6LS	IEFXK000
	IEFXJX5A			IEFXJX5A
	IEFV15XL			IEFV15XL
				IEFXJ000
IEFXJMSG	MSSYS	IEFXJMSG	IEFVM6LS	IEFXK000
	MSRCV			IEFXJX5A
	MSOFF			IEFV15XL
				IEFXJ000
IEFXKFAK	IEFXKFAK	IEFXK000	IEFWD000	IEFW41SD
IEFXKIMP	IEFXK000	IEFXK000	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
IEFXKMSG	IEFXKMSG	IEFXKMSG	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
IEFXTDMY	IEFXTDMY	IEFXTDMY	IEFWD000	IEFW41SD
IEFXTMSG	RESMSG	IEFXTMSG	IEFWD000	IEFW41SD
	MESGDSDV			
	MESGHEAD			
	MESGDDEV			
	PRIAMSG			
	JOBCAN			
	WTORREP			
	RESUNIT			
IEFXT00D	WDEXIT	IEFXT000	IEFWD000	IEFW41SD
IEFXT002	IEFXT002	IEFXT002	IEFWD000	IEFW41SD
	XTTEA0			
	XTTEA1			
	XTTEB3			
	XTTRDJ			
IEFXT003	XUUB00	IEFXT003	IEFWD000	IEFW41SD
	XUUH06			

(Continued)

Assembly Module (MODLIB) Name	Entry Point Name	CSECT Name	Load Module Name	Alias Name
	IEFXT003			
IEFXVMSG	IEFXVMSG	IEFXVMSG	IEFXVAVR	IEFXV001
IEFXVNSL	IEFXVNSL	IEFXVNSL	IEFXVAVR	IEFXV001
IEFXV001	IEFXV001	IEFXV001	IEFXVAVR	IEFXV001
IEFXV002	IEFXV002	IEFXV002	IEFXVAVR	IEFXV001
IEFXV003	IEFXV003	IEFXV003	IEFXVAVR	IEFXV001
IEFX300A	X33B42	IEFX3000	IEFXVAVR	IEFXV001
	IEFX3000		IEFWA000	IEFWC000
IEFX5000	IEFX5000	IEFX5000	IEFWA000	IEFWC000
	X55D3G			
	X55C86			
	XIIB32			
IEFYNIMP	IEFYN	IEFYN	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFYNMSG	IEFYNMSG	IEFYNMSG	IEFSD061	IEFW42SD
	STRMSG01			IEFSD064
				IEFSD104
IEFYPJB3	IEFYP	IEFYP	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFYPMSG	YPPMSG1	IEFYPMSG	IEFSD061	IEFW42SD
	IEFYPMSG			IEFSD104
	YPPMSG2			IEFSD064
IEFYRCDS	IEFYRCDS	IEFSD572	IEFSD061	IEFW42SD
		IEFYRCDS		IEFSD064
				IEFSD104
IEFYSVMS	IEFYS	IEFYS	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
			IEFWD000	IEFW41SD
IEFZGJBL	ZP0GM	IEFZGJ	IEFSD061	
IEFZGST1	IEFZG	IEFZG	IEFSD061	IEFW42SD
	ZG0J8			IEFSD064
	ZG0J5			IEFSD104
IEFZGST2	ZGOK09	IEFZG2	IEFSD06	IEFW42SD
	IEFZG2			IEFSD064
	ZPOQMGR1			IEFSD104
	ZPOC10			
	ZOOE10			
	ZOOA1			
IEFZHMSG	XPS631	IEFZH	IEFSD061	IEFW42SD
	ZK0E1			IEFSD064
	ZK0D1			IEFSD104
	ZG0E60			
	IEFZH			
IEF060SD	IEFSD060	IEFSD060	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEF061SD	IEFSD061	IEFSD061	IEFSD062	IEFV4221
			IEFSD060	
IEF064SD	IEFSD064	IEFSD064	IEFSD062	IEFV4221
IEF078SD	IEFSD078	IEFSD078	IEFSD080	IEFSD079
				IEEVWTR1
IEF079SD	IEFSD079	IEFSD079	IEFSD078	
IEF300SD	IEFSD300	IEFSD300	IEFSQINT	
IEF304SD	IEFSD304	IEFSD304	IEFSQINT	
IEF41FAK	IEFW2FAK	IEFW41SD	IEFMCVOL	IEFCVOL3
	IEFW1FAK			IEFCVOL2
	IEFW41SD			IEFCVOL1
			IEFW21SD	IEFVMCVL
				IEFVM1
				IEFXA
			IEFWA000	IEFWC000
IEZDCODE	IEZDCODE	IEZDCODE	IEFVMF	
			IEFVINA	
IEZNCODE	IEZNCODE	IEZNCODE	IEFVMB	
			IEFVINA	
IGC6103D	IGC6103D	IGC6103D	IGC6103D	
IGC6203D	IGC6203D	IGC6203D	IGC6203D	
IGF08501	IGF08501	IGF08501	IGF08501	
IGF08502	IGF08502	IGF08502	IGF08502	
IGF2603D	IGF2603D	IGF2603D	IGF2603D	
IGF29601	IGF29601	IGF29601	IGF29601	
IGF29701	IGF29701	IGF29701	IGF29701	
IGF55301	IGF55301	IGF55301	IGF55301	
IKJ5803D	IKJ5803D	IKJ5803D	IGC5803D	
IKJNULL	IKJNULL	IKJNULL	IGC5803D	

(Continued)

Entry Point Name	Assembly Module (MODLIB) Name	CSECT Name	Load Module Name	Alias Name
CNVRT	IEFCNVRT	IEFCNVRT	IEFVRR1 IEFLOCDQ	 LOC LOCCAN LOCDQ
IEECVCTI	IEECVINT	IEECVCTI	IEECVCTI	
IEELWAIT	IEELWAIT	IEELWAIT	IEELWAIT	
IEEPALTR	IEEPALTR	IEEPSTP1	IEEPALTR	
IEEPDISC	IEEPDISC	IEEPDISC	IEEPDISC	
IEEPPRES	IEEPPRES	IEEPPRES	IEEPPRES	
IEEPRTN2	IEEPRTN2	IEEPRTN2	IEEPRTN	
IEEPRWI2	IEEPRWI2	IEEPRWI2	IEEPRWI2	
IEEPSN	IEEPSN	IEEPSN	IEEVRJCL	IEEPSN
IEESD562	IEESD562	IEEDS56	IEEVDNX1	IEESD562
IEESD563	IEESD563	IEESD563	IEESD563	
IEESD564	IEESD564	IEESD564	IEESD564	
IEESD565	IEESD565	IEESD565	IEESD565	
IEESD575	IEESD575	IEESD575	IEESD575	IEESD581
IEESD576	IEESD576	IEESD576	IEESD576	IEESD580
IEESD577	IEESD577	IEESD577	IEESD577	
IEESD578	IEESD578	IEESD578	IEESD578	
IEESD579	IEESD579	IEEMSWTO IEESD579	IEESD579	
IEESD580	IEESD580	IEESD580	IEESD576	IEESD580
IEESD581	IEESD581	IEESD581	IEESD575	IEESD581
IEESD582	IEESD582	IEESD582	IEESD582	IEESD82A
IEESD583	IEESD583	IEESD583	IEESD583	
IEESD584	IEESD584	IEESD584	IEESD584	
IEESD82A	IEESD582	IEESD582	IEESD582	IEESD82A
IEESMFIT	IEESMFIT	IEESMFIT	IEESMFIT	IEESMF14
IEESMFIO	IEESMFI3	IEESMFI3	IEESMFI3	IEESMFMS IEESMFIO
IEESMFI2	IEESMFI2	IEESMFI2	IEESMFI2	
IEESMFI3	IEESMFI3	IEESMFI3	IEESMFI3	IEESMFIO IEESMFMS
IEESMFI4	IEESMFIT	IEESMFIT	IEESMFIT	IEESMFI4
IEESMFMS	IEESMFI3	IEESMFI3	IEESMFI3	IEESMFMS IEESMFI0
IEESMFOI	IEESMFOI	IEESMFOI	IEESMFOI	
IEESMFOP	IEESMFOP	IEESMFOP	IGC0108C	
IEESMFWR	IEESMFWR	IEESMFWR	IEESMFWR	
IEESMF8C	IEESMF8C	IGC0083	IGC0008C	
IEEVDRGN	IEEVDRGN	IEEVDRGN	IEEVDSP1	
IEEVDSP1	IEEVDSP1	IEEVDSP1	IEEVDSP1	
IEEVICLR	IEEVICLR	IEEVICLR	IEEVICLR	IEFICR
IEEVICTL	IEEVICTL	IEEVICTL	IEEVICTL	IEEVIC
IEEVIPL	IEEVIPL	IEEVIPL	IEEVIPL	
IEEVJCL	IEEVJCL	IEEVJCL	IEEVMNT1 IEEVSTAR	
IEEVLDSP	IEEVLDSP	IEEVLDSP	IEEVLDSP	
IEEVLIN1	IEEVLIN1	IEEVLIN1	IEEVLIN	
IEEVLOUT	IEEVLOUT	IEEVLOUT	IEEVLOUT	
IEEVMNT1	IEEVMNT1	IEEVMNT1	IEEVMNT1	
IEEVMNT2	IEEVMNT2	IEEVMNT2	IEEVMNT2	
IEEVOMSG	IEEVOMSG	IEEVOMSG	IEEVOMSG	
IEEVPRES	IEEVPRES	IEEVPRES	IEEVPRES	
IEEVRC	IEEVRC	IEEVRC	IEEVRC	
IEEVRCTL	IEEVRCTL	IEEVLNKT IEEVRJCL	IEEVRCTL	

(Continued)

Entry Point Name	Assembly Module (MODLIB) Name	CSECT Name	Load Module Name	Alias Name
		IEEPSN IEEVRCTL		
IEEVRFRX	IEEVRFRX	IEEVRFRX	IEEVLIN IEECVCTI	
IEEVRJCL	IEEVRJCL	IEEVRJCL	IEEVRJCL	IEEPSN
IEEVSIPL	IEEVSIPL	IEEVSIPL	IEEVSIPL	
IEEVSMSG	IEEVSMSG	IEEVSMSG	IEEVICTL IEEVMNT2 IEFIRC	
IEEVSTAR	IEEVSTAR	IEEVJCL IEEVSTAR	IEEVSTAR	
IEEVWAIT	IEEVWAIT	IEEVWAIT	IEEVWAIT	IEEVWDAR
IEEXEDNA	IEEXEDNA	IEEXEDNA	IEEXEDNA	
IEE0003D	IEE0003D	IEE0003D	IGC0003D	
IEE00110	IEE00110	IEE00110	IGC00110	
IEE0303D	IEE0303D	IEE0303D	IGC0303D	
IEE0503D	IEE0503D	IEE0503D	IGC0503D IEE0503D	
IEE0603D	IEE0603D	IEE0603D	IGC0603D	
IEE0703D	IEE0703D	IEE0703D	IGC0703D	
IEE0903D	IEE0903D	IEE0903D	IGC0903D	
IEE1A03D	IEE1A03D	IEE1A03D	IGC1203D	
IEE1B03D	IEE1B03D	IEE1B03D	IGC1B03D	
IEE10110	IEE10110	IEE10110	IGC10110	
IEE1103D	IEE1103D	IEE1103D	IGC1103D	
IEE11110	IEE11110	IEE11110	IGC11110	
IEE1203D	IEE1203D	IEE1203D	IGC1203D	
IEE12110	IEE12110	IEE12110	IGC12110	
IEE1403D	IEE1403D	IEE1403D	IGC1403D	
IEE1603D	IEE1603D	IEE1603D	IGC1603D	
IEE20110	IEE20110	IEE20110	IGC20110	
IEE2103D	IEE2103D	IEE2103D	IGG2103D	
IEE21110	IEE21110	IEE21110	IGC21110	
IEE22110	IEE22110	IEE22110	IGC22110	
IEE2303D	IEE2303D	IEE2303D	IGC2303D	
IEE23110	IEE23110	IEE23110	IGC23110	
IEE2903D	IEE2903D	IEE2903D	IGC2903D	
IEE30110	IEE30110	IEE30110	IGC30110	
IEE3103D	IEE3103D	IEE3103D	IGC3103D	
IEE31110	IEE31110	IEE31110	IGC31110	
IEE3203D	IEE3203D	IEE3203D	IGC3203D	
IEE32110	IEE32110	IEE32110	IGC32110	
IEE3303D	IEE3303D	IEE3303D	IGC3303D	
IEE3503D	IEE3503D	IEE3503D	IGC3503D	
IEE3603D	IEE3603D	IEE3603D	IGC3603D	
IEE3703D	IEE3703D	IEE3703D	IGC3703D	
IEE3803D	IEE3803D	IEE3803D	IGC3803D	
IEE40110	IEE40110	IEE40110	IGC40110	
IEE4103D	IEE4103D	IEE4103D	IGC4103D	
IEE4203D	IEE4203D	IEE4203D	IGC4203D	
IEE4303D	IEE4303D	IEE4303D	IGC4303D	
IEE4403D	IEE4403D	IEE4403D	IGC4403D	
IEE4503D	IEE4503D	IEE4503D	IGC4503D	
IEE4603D	IEE4603D	IEE4603D	IGC4603D	
IEE4703D	IEE4703D	IEE4703D	IGC4703D	
IEE4803D	IEE4803D	IEE4803D	IGC4803D	
IEE4903D	IEE4903D	IEE4903D	IGC4903D	
IEE50110	IEE50110	IEE50110	IGC50110	

(Continued)

Entry Point Name	Assembly Module (MODLIB) Name	CSECT Name	Load Module Name	Alias Name
IEE5103D	IEE5103D	IEE5103D	IGC5103D	
IEE5203D	IEE5203D	IEE5203D	IGC5203D	
IEE5303D	IEE5303D	IEE5303D	IGC5303D	
IEE5403D	IEE5403D	IEE5403D	IGC5403D	
IEE5503D	IEE5503D	IEE5503D	IGC5503D	
IEE5603D	IEE5603D	IEE5603D	IGC5603D	
IEE5903D	IEE5903D	IEE5903D	IGC5903D	
IEE60110	IEE60110	IEE60110	IGC60110	
IEE6303D	IEE6303D	IEE6303D	IGC6303D	
IEE6403D	IEE6403D	IEE6403D	IGC6403D	
IEE6503D	IEE6503D	IEE6503D	IGC6503D	
IEE6603D	IEE6603D	IEE6603D	IGC6603D	
IEE6703D	IEE6703D	IEE6703D	IGC6703D	
IEE6803D	IEE6803D	IEE6803D	IGC6803D	
IEE6903D	IEE6903D	IEE6903D	IGC6903D	
IEE7103D	IEE7103D	IEE7103D	IGC7103D	
IEE7203D	IEE7203D	IEE7203D	IGC7203D	
IEE7303D	IEE7303D	IEE7303D	IGC7303D	
IEE7503D	IEE7503D	IEE7503D	IGC7503D	
IEE7603D	IEE7603D	IEE7603D	IGC7603D	
IEE7703D	IEE7703D	IEE7703D	IGC7703D	
IEE7803D	IEE7803D	IEE7803D	IGC7803D	
IEE7903D	IEE7903D	IEE7903D	IGC7903D	
IEE8503D	IEE8503D	IEE8503D	IGC8503D	
IEE8603D	IEE8603D	IEE8603D	IGC8603D	
IEFACT	IEFACT	IEFACT	IEFVHA	IEFVHCB
IEFACTLK	IEFSMFLK	IEFACTLK	IEFSD061	IEFW42SD
				IEFSD065
				IEFSD104
	IEFACTFK	IEFACTLK	IEFW21SD	IEFVM1
				IEFXA
				IEFVMCVL
IEFACTRT	IEFACTRT	IEFACTRT	IEFSD061	IEFSD065
				IEFSD104
				IEFW42SD
			IEFW21SD	IEFVMCVL
				IEFVM1
				IEFXA
IEFBR14	IEFBR14	IEFBR14	IEFBR14	
IEFCVOL1	IEFMCVOL	IEFCVOL1	IEFMCVOL	IEFCVOL1
		IEFCVOL2		IEFCVOL2
		IEFCVOL3		IEFCVOL3
	IEFCVFAK	IEFCVOL1	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
			IFFVM6LS	IEFV15XL
				IEFXJX5A
				IEFXJ000
				IEFXK000
			IFFWD000	IEFW41SD
			IEFXVAVR	IEFXV001
IEFCVOL2	IEFCVFAK	IEFCVOL1	IEFXVAVR	IEFXV001
			IEFWD000	IEFW41SD
			IEFVM6LS	IEFXJX5A
				IEFV15XL
				IEFXJ000
				IEFXK000
			IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
	IEFMCVOL	IEFCVOL1	IEFMCVOL	IEFCVOL2
		IEFCVOL2		IEFCVOL1
		IEFCVOL3		IEFCVOL3

(Continued)

Entry Point Name	Assembly Module (MODLIB) Name	CSECT Name	Load Module Name	Alias Name
IEFCVOL3	IEFMCVOL	IEFCVOL3	IEFMCVOL	IEFCVOL3
		IEFCVOL2		IEFCVOL2
		IEFCVOL1		IEFCVOL1
	IEFCVFAK	IEFCVOL1	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
			IEFVM6LS	IEFXJ000
				IEFXJX5A
				IEFXK000
				IEFV15XL
			IEFWD000	IEFW41SD
			IEFXVAVR	IEFXV001
IEFDEVPT	IEFDEVPT	IEFDEVPT	IEFWA000	IEFWC000
			IEFW21SD	IEFVMCVL
				IEFVM1
				IEFXA
IEFDSDRP	IEFDSDRP	IEFDSDRP	IEFDSDRP	
IEFDSLST	IEFDSLST	IEFDSLST	IEFSD061	IEFSD104
				IEFSD064
				IEFW42SD
IEFDSOAL	IEFDSOAL	IEFDSOAL	IEFDSOAL	
IEFDSOCP	IEFDSOCP	IEFDSOCP	IEFDSO	
IEFDSOCR	IEFDSOCR	IEFDSOCR	IEFDSOCR	
IEFDSOFB	IEFDSOFB	IEFDSOFB	IEFDSOFB	
IEFDSOLP	IEFDSOLP	IEFDSOLP	IEFDSOLP	
IEFDSOSM	IEFDSOSM	IEFDSOSM	IEFDSOSM	
IEFDSOWR	IEFDSOWR	IEFDSOWR	IEFDSOWR	
IEFDSTBL	IEFDSTBL	IEFDSTBL	IEFSD061	
				IEFSD064
				IEFSD104
				IEFW42SD
IEFDSTRT	IEFDSTRT	IEFDSTRT	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFICR	IEEVICLR	IEEVICLR	IEEVICLR	IEFICR
IEFIDMPM	IEFIDMPM	IEFIDMPM	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFIDUMP	IEFIDUMP	IEFIDUMP	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFIIC	IEFIIC	IEFIIC	IEFIIC	
IEFIRC	IEFIRC	IEFIRC	IEFIRC	
IEFIRCB	IEFIRCB	IEFIRCB	IEFQMSSS	IEFIRCB
IEFK1MSG	IEFK1MSG	IEFK1MSG	IEEVPRES	
IEFORMAT	IEFORMAT	IEFORMAT	IEFSQINT	
IEFPRES	IEFPRES	IEFPRES	IEEVPRES	
IEFQAGST	IEFQAGST	IEFQAGST	IEFVMF	
			IEFVMB	
			IEFQMSSS	IEFIRCB
IEFQASGN	IEFQASGQ	IEFQASNM	IEFQMSSS	IEFIRCB
		IEFQASGN		
			IEFMCVOL	IEFCVOL1
				IEFCVOL2
				IEFCVOL3
			IEFVMB	
			IEFVMF	
			IEFSD304	
			IEFSD061	IEFSD064
				IEFSD104
				IEFW42SD
			IEFSD062	IEFV4221
			IEFWD000	IEFW41SD

(Continued)

Entry Point Name	Assembly Module (MODLIB) Name	CSECT Name	Load Module Name	Alias Name
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFW21SD	IEFVMCVL
				IEFVM1
				IEFXA
			IEFVRRC	IEFVRRCB
				IEFVRRCA
IEFQDELE	IEFQDELQ	IEFQDELE	IEFVRR3	IEFVR3AE
			IEFSD080	IEFSD079
				IEEVWTR1
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
				IEFV4221
			IEFSD304	
			IEFVMF	
			IEFVMD	
			IEFQDELE	
			IEESD576	IEESD580
IEFQINTZ	IEFQINTZ	IEFQINTZ	IEFQINTZ	
IEFQMDQ2	IEFQMDQQ	IEFQMDQ2	IEFQMDQQ	IEFQMDQ2
			IEFVMF	
			IEFSD061	IEFW41SD
				IEFSD064
				IEFSD104
				IEFV4221
			IEFSD080	IEFSD079
				IEEVWTR1
IEFQMNQ2	IEFQMNQQ	IEFQMNQ2	IEFSD061	IEFW42SD
			IEFSD068	
				IEFSD064
				IEFSD104
			IEFVMF	
			IEFVMB	
			IEFSD304	
			IEFQMSSS	IEFIRCB
			IEFQMNQ2	
			IEFDSOCR	
			IEFSD577	
			IEFSD579	
IEFQMRAW	IEFQMRAW	IEFQMRAW	IEFDSOCR	
		IEFQMWTO	IEFDSO	
			IEFDSDRP	
			IEFQMRAW	
			IEFQMSSS	IEFIRCB
			IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
			IEFSD304	
			IEFVMF	
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFSD068	
			IEFSD062	IEFV4221
			IEFSD086	IEF086SD
			IEFSD085	IEFSD071
				IEF085SD
				IEF850SD
			IEFVRRC	IEFVRRCB
				IEFVRRCA
			IEFW21SD	IEFVMCVL

(Continued)

Entry Point Name	Assembly Module (MODLIB) Name	CSECT Name	Load Module Name	Alias Name
				IEFXA
				IEFVM1
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFXVAVR	IEFXV001
			IEFWD000	IEFW41SD
IEFQMSSS	IEFQMLK1	IEFQMSSS	IEFXVAVR	IEFXV001
			IEFWD000	IEFW41SD
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
	IEFQBVMS	IEFQMSSS	IEFQMSSS	IEFIRCB
			IEFDSDRP	
			IEFVMF	
IEFQMUNC	IEFQMUNQ	IEFQMUNC	IEFQMUNC	
			IEFVMF	
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFSD080	IEEVWTR1
				IEFSD079
IEFRCLN1	IEFRCLN1	IEFRCLN1	IEFRCLN1	
IEFRCLN2	IEFRCLN2	IEFRCLN2	IEFRCLN2	
IEFRPREP	IEFRPREP	IEFRPREP	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFRSTRT	IEFRSTRT	IEFRSTRT	IEFRSTRT	IEFSMR
IEFSCAN	IEFSCAN	IEFSCAN	IEFWA000	IEFWC000
			IEFW21SD	IEFVMCVL
				IEFVM1
				IEFXA
IEFSDTTE	IEFSDTTE	IEFSDTTE	IEFSDTTE	
IEFSDXXX	IEFSDXXX	IEFSDXXX	IEFSDXXX	
IEFSDXYZ	IEFSDXYZ	IEFSDXYZ	IEFSDXYZ	
IEFSD0XX	IEFSD0XX	IEFSD0XX	IEFSD0XX	
IEFSD012	IEFVSD12	IEFSD012	IEFVHA	IEFVHCB
IEFSD055	IEFSD055	IEFSD055	IEFSQINT	
IEFSD060	IEFSD160	IEFSD60M	IEFSD060	
		IEFSD060		
	IEF060SD	IEFSD060	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFSD061	IEF061SD	IEFSD061	IEFSD062	IEFV4221
			IEFSD060	
	IEFDS161	IEFDSOAL	IEFSD061	IEFW42SD
		QMTMSD		IEFSD064
		IEFSD061		IEFSD104
IEFSD062	IEFSD162	IEFSD062	IEFSD062	IEFV4221
IEFSD063	IEFSD103	IEFSD063	IEFSD062	IEFV4221
IEFSD064	IEFSD164	IEFSD064	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104

(Continued)

Entry Point Name	Assembly Module (MODLIB) Name	CSECT Name	Load Module Name	Alias Name
IEFSD066	IEFSD166	IEFSD066	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFSD068	IEFSD168	IEFSD068	IEFSD068	IEFW42SD
				IEFSD064
				IEFSD104
			IEFDSOCR	
IEFSD070	IEFSD070	IEFSD070	IEFSD070	
IEFSD071	IEFSD171	IEFSD71M	IEFSD085	IEF850SD
		IEFSD071		IEF085SD
				IEFSD071
IEFSD078	IEF078SD	IEFSD078	IEFSD080	IEFSD079
				IEEVWTR1
	IEFSD078	IEFSD078	IEFSD078	
IEFSD079	IEF079SD	IEFSD079	IEFSD078	
	IEFSD079	IEFSD079	IEFSD080	IEFSD079
				IEEVWTR1
IEFSD080	IEFSD080	IEFSD080	IEFSD080	IEFSD079
				IEEVWTR1
IEFSD081	IEFSD081	IEFSD081	IEFSD080	IEFSD079
				IEEVWTR1
IEFSD082	IEFSD082	IEFSD082	IEFSD080	IEFSD079
				IEEVWTR1
IEFSD083	IEFSD083	IEFSD83M	IEFSD080	IEFSD079
		IEFSD083		IEEVWTR1
IEFSD084	IEFSD084	IEFSD084	IEFSD080	IEFSD079
				IEEVWTR1
IEFSD085	IEFSD085	IEFSD085	IEFSD085	IEF085SD
				IEFSD071
				IEF850SD
IEFSD086	IEFSD086	IEFSD086	IEFSD086	IEF086SD
IEFSD087	IEFSD087	IEFSD87M	IEFSD087	
		IEFSD087		
IEFSD088	IEFSD088	IEFSD088	IEFSD087	
			IEFSD094	
			IEFSD086	IEF086SD
IEFSD089	IEFSD089	IEFSD89M	IEFSD086	IEF086SD
		IEFSD089	IEFSD094	
			IEFSD087	
IEFSD090	IEFVSD13	IEFSD090	IEFVHA	IEFVHCB
IEFSD094	IEFSD094	IEFSD094	IEFSD094	
IEFSD095	IEFSD095	IEFSD095	IEFSD094	
IEFSD096	IEFSD096	IEFSD096	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFWD000	IEFW41SD
IEFSD097	IEFSD097	IEFSD097	IEFWD000	IEFW41SD
IEFSD101	IEFSD101	IEFSD101	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFSD102	IEFSD102	IEFSD102	IEFSD102	
IEFSD104	IEFSD104	IEFSD104	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFSD105	IEFSD105	IEFSD105	IEFSD105	
IEFSD110	IEFSD110	IEFSD110	IEFSD062	IEFV4221
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFVMF	
IEFSD111	IEFSD111	IEFSD111	IEFVMF	
		CHFILDRW	IEFSD061	IEFW42SD

(Continued)

Entry Point Name	Assembly Module (MODLIB) Name	CSECT Name	Load Module Name	Alias Name
				IEFSD064
				IEFSD104
			IEFSD068	
			IEFSD062	IEFW4221
			IEFWD000	IEFW41SD
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
			IEFXVAVR	IEFXV001
			IEFDSOCR	
			IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
IEFSD112	IEFSD112	IEFSD112	IEFDSOCR	
			IEFVRR3	IEFVR3AE
			IEFSD062	IEFV4221
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFSD112	
			IEFVMF	
IEFSD18	IEFSD180	IEFSD180	IEFW21SD	IEEVMCVL
				IEFXA
				IEFVM1
IEFSD263	IEFSD263	IEFSD263	IEFSD263	
IEFSD300	IEFSD300	IEFSD300	IEFSD300	
	IEF300SD	IEFSD300	IEFSQINT	
IEFSD301	IEFSD301	IEFSD301	IEFSD300	
IEFSD302	IEFSD302	IEFSD302	IEFSD300	
IEFSD303	IEFSD303	IEFSD303	IEFSD300	
IEFSD304	IEFSD304	IEFSD304	IEFSD304	
	IEF304SD	IEFSD304	IEFSQINT	
IEFSD305	IEFSD305	IEFSD305	IEFSD304	
IEFSD308	IEFSD308	IEFSD308	IEFSD308	
IEFSD310	IEFSD310	IEFSD310	IEFSD304	
			IEFSD300	
IEFSD311	IEFSD311	IEFSD311	IEFSQINT	
IEFSD312	IEFSD312	IEFSD312	IEFSD304	
			IEFSD300	
IEFSD514	IEFSD514	IEFSD514	IEFSD304	
			IEFSD062	IEFV4221
			IEFVRRC	IEFVRRCB
				IEFVRRCA
IEFSD551	IEFSD551	IEFV15XL	IEFWA000	IEFWC000
			IEFXVAVR	IEFXV001
IEFSD552	IEFSD552	IEFXJX5A	IEFWA000	IEFWC000
			IEFXVAVR	IEFXV001
IEFSGOPT	IEFSGOPT	IEFSGOPT	IEFWA000	IEFWC000
IEFSMFAT	IEFSMFAT	IEFSMFAT	IEFSD263	
IEFSMFIE	IEFSMFIE	IEFSMFIE	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFSMFWI	IEFSMFWI	IEFSMFWI	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFSMR	IEFRSTRT	IEFRSTRT	IEFRSTRT	IEFSMR
IEFUCBL	IEFWA000	IEFWA7	IEFW21SD	IEFVMCVL
		IEFWA002		IEFVM1
				IEFXA

(Continued)

Entry Point Name	Assembly Module (MODLIB) Name	CSECT Name	Load Module Name	Alias Name
IEFUJI	IEFUJI	IEFUJI	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFUJV	IEFUJV	IEFUJV	IEFUJV	
IEFUSI	IEFUSI	IEFUSI	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFUSO	IEFUSO	IEFUSO	IEFSD263	
IEFUTL	IEFUTL	IEFUTL	IEFSD263	
IEFVAWAT	IEFSD195	IEFSD095	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFWD000	IEFW41SD
				IEFW41SD
IEFVDA	IEFVDA	IEFVDA	IEFVHA	IEFVHCB
IEFVDBSD	IEFVDBSD	IEFVDBSD	IEFVHA	IEFVHCB
IEFVEA	IEFVEA	IEFVEA	IEFVHA	IEFVHCB
IEFVFA	IEFVFA	IEFVFA	IEFVHA	IEFVHCB
IEFVFB	IEFVFB	IEFVFB	IEFVHA	IEFVHCB
IEFVGI	IEFVGI	IEFVGI	IEFVHA	IEFVHCB
IEFVGK	IEFVGK	IEFVGK	IEFVHA	IEFVHCB
IEFVGM	IEFVGM	IERFVGM	IEFVHA	IEFVHCB
			IEFINA	
IEFVGS	IEFVGS	IEFVGS	IEFVHA	IEFVHCB
IEFVGT	IEFVGT	IEFVGT	IEFVHA	IEFVHCB
IEFVHA	IEFVHA	IEFVHA	IEFVHA	IEFVHCB
IEFVHAA	IEFVHAA	IEFVHAA	IEFVHA	IEFVHCB
IEFVHB	IEFVHB	IEFVBH	IEFVHA	IEFVHCB
IEFVHC	IEFVHC	IEFVHC	IEFVHA	IEFVHCB
IEFVHCB	IEFVHCB	IEFVHCB	IEFVHA	IEFVHCB
IEFVHE	IEFVHE	IEFVHE	IEFVHA	IEFVHCB
IEFVHEB	IEFVHEB	IEFVHEB	IEFVHA	IEFVHCB
IEFVHEC	IEFVHEC	IEFVHEC	IEFVHA	IEFVHCB
IEFVHF	IEFVHF	IEFVHF	IEFVHA	IEFVHCB
IEFVHG	IEFVHG	IEFVHG	IEFVHA	IEFVHCB
IEFVHH	IEFVHH	IEFVHH	IEFVHA	IEFVHCB
IEFVHHB	IEFVHHB	IEFVHHB	IEFVHA	IEFVHCB
IEFVHL	IEFVHL	IEFVHL	IEFVHA	IEFVHCB
IEFVHM	IEFVHM	IEFVHM	IEFVHA	IEFVHCB
IEFVHN	IEFVHN	IEFVHN	IEFVHN	
IEFVHQ	IEFVHQ	IEFVHQ	IEFVHA	IEFVHCB
			IEFVINA	
IEFVHR	IEFVHR	IEFVHR	IEFVHA	IEFVHCB
			IEFVINA	
IEFVH1	IEFVH1	IEFVH1	IEFVH1	
IEFVH2	IEFVH2	IEFVH2	IEFVH1	
IEFVINA	IEFVINA	IEFVINA	IEFVINA	
IEFVINB	IEFVINB	IEFVINB	IEFVINA	
			IEFVINB	
IEFVINC	IEFVINC	IEFVINC	IEFVINA	
IEFVIND	IEFVIND	IEFVIND	IEFVHA	IEFVHCB
IEFVINE	IEFVINE	IEFVINE	IEFVINA	
IEFVJ	IEFVJIMP	IEFVJ	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFVJA	IEFVJA	IEFVJA	IEFVHA	IEFVHCB
IEFVJMSG	IEFVJMSG	IEFVJMSG	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFVK	IEFVKIMP	IEFVK	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1

(Continued)

Entry Point Name	Assembly Module (MODLIB) Name	CSECT Name	Load Module Name	Alias Name
IEFVKG	IEFVKG	IEFVKG	IEFVHA	IEFVHCB
IEFVKMJ1	IEFVKMSG	IEFVKMSG	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFVM	IEFVMLS1	IEFVM1	IEFW21SD	IEFVMCVL
		IEFVMQMI		IEFXA
		IEFVMPDQ		IEFVM1
		IEFVMVTE		
IEFVMA	IEFVMA	IEFVMA	IEFVMA	
IEFVMB	IEFVMB	IEFVMB	IEFVMB	
IEFVMC	IEFVMC	IEFVMC	IEFVMC	
IEFVMCVL	IEFVMLS1	IEFVM1	IEFW21SD	IEFVMCVL
		IEFVMQMI		IEFXA
		IEFVMPDQ		IEFVM1
		IEFVMVTE		
	IEFVMFAK	IEFVMCVL	IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
IEFVMD	IEFVMD	IEFVMD	IEFVMD	
IEFVME.	IEFVME	IEFVME	IEFVMF	
IEFVMF	IEFVMF	IEFVMF	IEFVMF	
IEFVMG	IEFVMG	IEFVMG	IEFVMF	
IEFVMH	IEFVMH	IEFVMH	IEFVMF	
IEFVMLK5	IEFVMLK5	IEFVM6	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFVMQMI	IEFVMLS1	IEFVM1	IEFW21SD	IEFVMCVL
		IEFVMQMI		IEFXA
		IEFVMPDQ		IEFVM1
		IEFVMVTE		
IEFVMSGR	IEFVMLS6	IEFVM6	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
IEFVM1	IEFVMMS1	IEFVM1	IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFWD000	IEFW41SD
IEFVM7	IEFVMLS7	IEFVM7	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
IEFVRRC	IEFVRRC	IEFVRRC	IEFVRRC	IEFVRRCB
				IEFVRRCA
IEFVRRCA	IEFVRRC	IEFVRRC	IEFVRRC	IEFVRRCB
				IEFVRRCA
IEFVRRCB	IEFVRRC	IEFVRRC	IEFVRRC	IEFVRRCB
				IEFVRRCA
IEFVRR1	IEFVRR1	IEFVRR1	IEFVRR1	
IEFVRR2	IEFVRR2	IEFVRR2	IEFVRR2	IEFVR2AE
IEFVRR3	IEFVRR3	IEFVRR3	IEFVRR3	IEFVR3AE
IEFVSDRA	IEFVSDRA	IEFVSDRA	IEFSD068	IEFW42SD
				IEFSD064

(Continued)

Entry Point Name	Assembly Module (MODLIB) Name	CSECT Name	Load Module Name	Alias Name
				IEFSD104
			IEFSD304	
IEFVSDRD	IEFVSDRD	IEFVSDRD	IEFSD304	
IEFVSMSG	IEFVSMSG	IEFVSMSG	IEFVH1	
IEFW15XL	IEFXJIMP	IEFXJ000	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
IEFV2AE	IEFVRR2	IEFVRR2	IEFVRR2	IEFVR2AE
IEFV3AE	IEFVRR3	IEFVRR3	IEFVRR3	IEFVR3AE
IEFWAD	IEFWAD	IEFWAD	IEFW21SD	IEFVMCVL
				IEFVM1
				IEFXA
IEFWA000	IEFWA000	IEFWA002	IEFWA000	IEFWC000
IEFWC000	IEFWCFAK	IEFWC000	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
			IEFXVAVR	IEFXV001
	IEFWCIMP	IEFWC000	IEFWA000	IEFWC000
IEFWD000	IEFWD000	IEFWD000	IEFWD000	IEFW41SD
			IEFXVAVR	IEFXV001
	IEFWDFAK	IEFWD000	IEFWA000	IEFWC000
IEFWD001	IEFWD001	IEFWD001	IEFXVAVR	IEFXV001
			IEFWD000	IEFW41SD
IEFWEXTA	IEFWEXTA	IEFWEXTA	IEFWD000	IEFW41SD
IEFWSTRT	IEFWSTRT	IEFWSTRT	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFWSWIT	IEFWSWIN	IEFWSWIT	IEFWA000	IEFWC000
IEFWTERM	IEFWTERM	IEFWTERM	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFW1FAK	IEFSD42Q	IEFW41SD	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
	IEF41FAK	IEFW41SD	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
			IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
			IEFWA000	IEFWC000
	IEFSD41Q	IEFW41SD	IEFWD000	IEFW41SD
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
IEFW2FAK	IEF41FAK	IEFW41SD	IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
			IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
			IEFWA000	IEFWC000
	IEFSD41Q	IEFW41SD	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFWD000	IEFW41SD
	IEFSD42Q	IEFW42SD	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFW21SD	IEFSD21Q	IEFW21SD	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFW22SD	IEFSD22Q	IEFW22SD	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFW31SD	IEFSD31Q	IEFW31SD	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFW41SD	IEFSD42Q	IEFW41SD	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
	IEFSD41Q	IEFW41SD	IEFVM6LS	IEFXK000
				IEFXJX5A
				IEFXJ000
				IEFV15XL
			IEFWD000	IEFW41SD
	IEF41FAK	IEFW41SD	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
			IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
			IEFWA000	IEFWC000
IEFXA	IEFXCSSS	IEXAB00	IEFW21SD	IEFXMCVL
		IEFXA		IEFVM1
				IEFXA
IEFXAFAK	IEFXAFAK	IEFXA	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
IEFXAMSG	IEFXAMSG	IEFXAMSG	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFXDPTH	IEFXDPTH	IEFXDPTH	IEFWA000	IEFWC000
IEFXH000	IEFXH000	IEFXH000	IEFWA000	IEFWC000
			IEFXVAVR	IEFXV001
IEFXJX5A	IEFXJIMP	IEFXJ000	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
IEFXJ000	IEFXJIMP	IEFXJ000	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
	IEFXJFAK	IEFXJ000	IEFXVAVR	IEFXV001
			IEFWA000	IEFWC000
			IEFWD000	IEFW41SD
IEFXKFAK	IEFXKFAK	IEFXK000	IEFWD000	IEFW41SD
IEFXKMSG	IEFXKMSG	IEFXKMSG	IEFMV6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
IEFXK000	IEFXKIMP	IEFXK000	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
IEFXTDMY	IEFXTDMY	IEFXTDMY	IEFWD000	IEFW41SD
IEFXT000	IEFXT00D	IEFXT000	IEFWD000	IEFW41SD
IEFXT002	IEFXT002	IEFXT002	IEFWD000	IEFW41SD
IEFXT003	IEFXT003	IEFXT003	IEFWD000	IEFW41SD
IEFXVMSG	IEFXVMSG	IEFXVMSG	IEFXVAVR	IEFXV001
IEFXVNSL	IEFXVNSL	IEFXVNSL	IEFXVAVR	IEFXV001
IEFXV001	IEFXV001	IEFXV001	IEFXVAVR	IEEXV001
	IEFAVFAK		IEFWA000	IEFWC000
IEFXV002	IEFXV002	IEFXV002	IEFXVAVR	IEFXV001

(Continued)

Entry Point Name	Assembly Module (MODLIB) Name	CSECT Name	Load Module Name	Alias Name
IEFXV003	IEFXV003	IEFXV003	IEFXVAVR	IEFXV001
IEFX3000	IEFX300A	IEFX3000	IEFXVAVR	IEFXV001
			IEFWA000	IEFWC000
IEFX5000	IEFX5000	IEFX5000	IEFWA000	IEFWC000
			IEFXVAVR	IEFXV001
IEFYN	IEFYNIMP	IEFYN	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFYNMSG	IEFYNMSG	IEFYNMSG	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFYP	IEFYPJB3	IEFYP	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFYPMSG	IEFYPMSG	IEFYPMSG	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFYRCDS	IEFYRCDS	IEFSD572	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFYS	IEFYSVMS	IEFYS	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
			IEFWD000	IEFW41SD
IEFZG	IEFZGST1	IEFZG	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFZG2	IEFZGST2	IEFZG2	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFZH	IEFZHMSG	IEFZH	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEF085SD	IEFSD085	IEFSD85M	IEFSD085	IEF850SD
		IEFSD085		IEFSD071
				IEF085SD
IEF086SD	IEFSD086	IEFSD86M	IEFSD086	IEF086SD
		IEFSD086		
IEF850SD	IEFSD085	IEFSD85M	IEFSD085	IEF850SD
		IEFSD085		IEFSD071
				IEF085SD
IEZDCODE	IEZDCODE	IEZDCODE	IEFVMF	
			IEFVINA	
IEZNCODE	IEZNCODE	IEZNCODE	IEFVMB	
			IEFVINA	
IGC0005B	IEFVSMBR	IGC0005B	IGC0005B	
IGC0203E	IEESMFAL	IGC0203E	IGC0208C	
	IEFWTP00	IGC0203E	IGC0203E	
IGC0303E	IEFWTP01	IGC0303E	IGC0303E	
IGC0403E	IEFWTP02	IGC0403E	IGC0403E	
IGC6103D	IGC6103D	IGC6103D	IGC6103D	
IGC6203D	IGC6203D	IGC6203D	IGC6203D	
IGF08501	IGF08501	IGF08501	IGF08501	
IGF08502	IGF08502	IGF08502	IGF08502	
IGF2603D	IGF2603D	IGF2603D	IGF2603D	
IGF29601	IGF29601	IGF29601	IGF29601	
IGF29701	IGF29701	IGF29701	IGF29701	

(Continued)

Entry Point Name	Assembly Module (MODLIB) Name	Csect Name	Load Module Name	Alias Name
IGF55301	IGF55301	IGF55301	IGF55301	
IKJNULL	IKJNULL	IKJNULL	IGC5803D	
IKJ5803D	IKJ5803D	IKJ5803D	IGC5803D	
JOBCAN	IEFXTMSG	IEFXTMSG	IEFWD000	IEFW41SD
LOC	IEFLOCDQ	IEFLOCDQ	IEFVRR1	
			IEFLOCDQ	LOCCAN
				LOC
				LOCDQ
LOCCAN	IEFLOCDQ	IEFLOCDQ	IEFLOCDQ	LOCCAN
				LOC
				LOCDQ
			IEFVRR1	
LOCDQ	IEFLOCDQ	IEFLOCDQ	IEFVRR1	
			IEFLOCDQ	LOCDQ
				LOCCAN
				LOC
MESGDDEV	IEFXTMSG	IEFXTMSG	IEFWD000	IEFW41SD
MESGDSDV	IEFXTMSG	IEFXTMSG	IEFWD000	IEFW41SD
MESGHEAD	IEFXTMSG	IEFXTMSG	IEFWD000	IEFW41SD
MSOFF	IEFXJMSG	IEFXJMSG	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
MSRCV	IEFXJMSG	IEFXMJSG	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
MSSYS	IEFXJMSG	IEFXJMSG	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
PRIAMSG	IEFXTMSG	IEFXTMSG	IEFWD000	IEFW41SD
RD	IEFRDWRT	IEFRDWRT	IEFVRR1	
			IEFLOCDQ	IOCCAN
				LOCDQ
				LOC
RESMSG	IEFXTMSG	IEFXTMSG	IEFWD000	IEFW41SD
RESUNIT	IEFXTMSG	IEFXTMSG	IEFWD000	IEFW41SD
SD304MG1	IEFSD312	IEFSD312	IEFSD304	
			IEFSD300	
SD304MG2	IEFSD312	IEFSD312	IEFSD300	
			IEFSD304	
SD305MG1	IEFSD312	IEFSD312	IEFSD304	
			IEFSD300	
SD55MSG1	IEFSD311	IEFSD311	IEFSQINT	
SD55MSG2	IEFSD311	IEFSD311	IEFSQINT	
SD55MSG3	IEFSD311	IEFSD311	IEFSQINT	
SPRINTER	IEFPRTXX	SPRINTER	IEFPRINT	IEFPRT
STRMSG01	IEFYNMSG	IEFYNMSG	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
UCBCONT	IEFSCAN	IEFSCAN	IEFW21SD	IEFVMCVL
				IEFVM1
				IEFXA
			IEFWA000	IEFWC000
UCBSCAN	IEFSCAN	IEFSCAN	IEFW21SD	IEFVMCVL
				IEFVM1
				IEFXA
			IEFWA000	IEFWC000
VM7000	IEFVMLS1	IEFVM1	IEFW21SD	IEFVMCVL
		IEFVMQMI		IEFXA
		IEFVMPDQ		IEFVM1
		IEFVMVTE		
VM7055	IEFVMLS1	IEFVM1	IEFW21SD	IEFVMCVL

(Continued)

Entry Point Name	Assembly Module (MODLIB) Name	CSECT Name	Load Module Name	Alias Name
		IEFVMQMI IEFVMPDQ IEFVMVTE		IEFXA IEFVM1
VM7055AA	IEFVMLS1	IEFVM1 IEFVMQMI IEFVMPDQ IEFVMVTE	IEFW21SD	IEFVMCVL IEFXA IEFVM1
VM7060	IEFVMLS1	IEFVM1 IEFVMQMI IEFVMPDQ IEFVMVTE	IEFW21SD	IEFVMCVL IEFXA IEFVM1
VM7070	IEFVMLS1	IEFVM1 IEFVMQMI IEFVMPDQ IEFVMVTE	IEFW21SD	IEFVMCVL IEFXA IEFVM1
VM7090	IEFVMLS1	IEFVM1 IEFVMQMI IEFVMPDQ IEFVMVTE	IEFW21SD	IEFVMCVL IEFXA IEFVM1
VM7100	IEFVM2LS	IEFVM2	IEFW21SD	IEFVMCVL IEFXA IEFVM1
VM7130	IEFVMLS1	IEFVM1 IEFVMQMI IEFVMPDQ IEFVMVTE	IEFW21SD	IEFVMCVL IEFXA IEFVM1
VM7150	IEFVM3LS	IEFVM3	IEFW21SD	IEFVMCVL IEFXA IEFVM1
VM7200	IEFVM4LS	IEFVM4	IEFW21SD	IEFVMCVL IEFXA IEFVM1
VM7300	IEFVM5LS	IEFVM5	IEFW21SD	IEFVMCVL IEFXA IEFVM1
VM7370	IEFVMLS1	IEFVM1 IEFVMQMI IEFVMPDQ IEFVMVTE	IEFW21SD	IEFVMCVL IEFXA IEFVM1
VM7600	IEFVM76	IEFVM76	IEFW21SD	IEFVMCVL IEFXA IEFVM1
VM7700	IEFVMLS1	IEFVM1 IEFVMQMI IEFVMPDQ IEFVMVTE	IEFW21SD	IEFVMCVL IEFXA IEFVM1
VM7742	IEFVMLS1	IEFVM1 IEFVMQMI IEFVMPDQ IEFVMVTE	IEFW21SD	IEFVMCVL IEFXA IEFVM1
VM7750	IEFVMLS1	IEFVM1 IEFVMQMI IEFVMPDQ IEFVMVTE	IEFW21SD	IEFVMCVL IEFXA IEFVM1
VM7850	IEFVMLS1	IEFVM1 IEFVMQMI IEFVMPDQ IEFVMVTE	IEFW21SD	IEFVMCVL IEFXA IEFVM1
VM7900	IEFVMLS1	IEFVM1 IEFVMQMI IEFVMPDQ IEFVMVTE	IEFW21SD	IEFVMCVL IEFXA IEFVM1
VM7950	IEFVMSL1	IEFVM1 IEFVMQMI	IEFW21SD	IEFVMCVL IEFXA

(Continued)

Entry Point Name	Assembly Module (MODLIB) Name	CSECT Name	Load Module Name	Alias Name
		IEFVMPDQ IEFVMVTE		IEFVM1
WDEXIT	IEFXT00D	IEFXT000	IEFWD000	IEFW41SD
WRT	IEFRDWRT	IEFRDWRT	IEFVRR1	
			IEFLOCDQ	LOCCAN LOCDQ LOC
WTORREP	IEFXTMSG	IEFXTMSG	IEFWD000	IEFW41SD
XIIB32	IEFX5000	IEFX5000	IEFWA000 IEFXVAVR	IEFWC000 IEFXV001
XPS631	IEFZHMSG	IEFZH	IEFSD061	IEFW42SD IEFSD064 IEFSD104
XTTBC0	IEFXT00D	IEFXT000	IEFWD000	IEFW41SD
XTTEA0	IEFXT002	IEFXT002	IEFWD000	IEFW41SD
XTTEA1	IEFXT002	IEFXT002	IEFWD000	IEFW41SD
XTTEB3	IEFXT002	IEFXT002	IEFWD000	IEFW41SD
XTTRDJ	IEFXT002	IEFXT002	IEFWD000	IEFW41SD
XUUB00	IEFXT003	IEFXT003	IEFWD000	IEFW41SD
XUUH06	IEFXT003	IEFXT003	IEFWD000	IEFW41SD
X33B42	IEFX300A	IEFX3000	IEFWA000 IEFXVAVR	IEFWC000 IEFXV001
X55C86	IEFX5000	IEFX5000	IEFWA000 IEFXVAVR	IEFWC000 IEFXV001
X55D3G	IEFX5000	IEFX5000	IEFXVAVR IEFWA000	IEFXV001 IEFWC000
YPPMSG1	IEFYPMSG	IEFYPMSG	IEFSD061	IEFW42SD IEFSD064 IEFSD104
YPPMSG2	IEFYRCDS IEFYPMSG	IEFYRCDS IEFYPMSG	IEFSD061	IEFW42SD IEFSD104 IEFSD064
ZGOK09	IEFZGST2	IEFZG2	IEFSD061	IEFW42SD IEFSD064 IEFSD104
ZG0E60	IEFZHMSG	IEFZH	IEFSD061	IEFW42SD IEFSD064 IEFSD104
ZG0J5	IEFZGST1	IEFZG	IEFSD061	IEFW42SD IEFSD064 IEFSD104
ZG078	IEFZGST1	IEFZG	IEFSD061	IEFW42SD IEFSD064 IEFSD104
ZK0D1	IEFZHMSG	IEFZH	IEFSD061	IEFW42SD IEFSD064 IEFSD104
ZK0E1	IEFZHMSG	IEFZH	IEFSD061	IEFW42SD IEFSD064 IEFSD104
ZOOA1	IEFZGST2	IEFZG2	IEFSD061	IEFW42SD IEFSD064 IEFSD104
ZOOE10	IEFZGST2	IEFZG2	IEFSD061	IEFW42SD IEFSD064 IEFSD104
ZP0C10	IEFZGST2	IEFZG2	IEFSD061	IEFW42SD IEFSD064 IEFSD104
ZPOQMGR1	IEFZGST2	IEFZG2	IEFSD061	IEFW42SD IEFSD064 IEFSD104
ZP0QM	IEFZGJBL	IEFZGJ	IEFSD061	

CSECT Name	Assembly Module (MODLIB) Name	Entry Point Name	Load Module Name	Alias Name
CHFILDRW	IEFSD111	IEFSD111	IEFW21SD	IEFVMCVL IEFVM1 IEFXA
CHLOADTB	CHLOADTB		IEFW21SD	IEFXA IEFVM1 IEFVMCVL
IEECVCTI	IEECVINT	IEECVCTI	IEECVCTI	
IEELWAIT	IEELWAIT	IEELWAIT	IEELWAIT	
IEEMSWTO	IEESD579	IEESD579	IEESD579	
IEEPDISC	IEEPDISC	IEEPDISC	IEEPDISC	
IEEPPRES	IEEPPRES	IEEPPRES	IEEPPRES	
IEEPRTN2	IEEPRTN2	IEEPRTN2	IEEPRTN	
IEEPRWI2	IEEPRWI2	IEEPRWI2	IEEPRWI2	
IEEPSN	IEEPSN	IEEPSN	IEEVRJCL	IEEPSN
IEEPSTP1	IEEPALTR	IEEPALTR	IEEPALTR	
IEESD562	IEESD562	IEESD562	IEEVDNX1	IEESD562
IEESD563	IEESD563	IEESD563	IEESD563	
IEESD564	IEESD564	IEESD564	IEESD564	
IEESD565	IEESD565	IEESD565	IEESD565	
IEESD575	IEESD575	IEESD575	IEESD575	IEESD581
IEESD576	IEESD576	IEESD576	IEESD576	IEESD580
IEESD577	IEESD577	IEESD577	IEESD577	
IEESD578	IEESD578	IEESD578	IEESD578	
IEESD579	IEESD579	IEESD579	IEESD579	
IEESD580	IEESD580	IEESD580	IEESD576	IEESD580
IEESD581	IEESD581	IEESD581 IEESMFI4	IEESD575	IEESD581
IEESD582	IEESD582	IEESD582 IEESD82A	IEESD582	IEESD82A
IEESD583	IEESD583	IEESD583	IEESD583	
IEESD584	IEESD584	IEESD584	IEESD584	
IEESMFI2	IEESMFI2	IEESMFI2	IEESMFI2	
IEESMFI3	IEESMFI3	IEESMFI0 IEESMFI3 IEESMFMS	IEESMFI3	IEESMFMS IEESMFI0
IEESMFOI	IEESMFOI	IEESMFOI	IEESMFOI	
IEESMFOP	IEESMFOP	IEESMFOP	IGC0108C	
IEESMFWR	IEESMFWR	IEESMFWR	IEESMFWR	
IEEVDRGN	IEEVDRGN	IEEVDRGN	IEEVDSP1	
IEEVDSP1	IEEVDSP1	IEEVDSP1	IEEVDSP1	
IEEVICLR	IEEVICLR	IEEVICLR IEFICR	IEEVICLR	IEFICR
IEEVICTL	IEEVICTL	IEEVICTL	IEEVICTL	IEEVIC
IEEVIPL	IEEVIPL	IEEVIPL	IEEVIPL	
IEEVJCL	IEEVJCL	IEEVJCL	IEEVMNT1 IEEVSTAR	
	IEEVSTAR	IEEVSTAR	IEEVSTAR	
IEEVLDSP	IEEVLDSP	IEEVLDSP	IEEVLDSP	
IEEVLIN1	IEEVLIN1	IEEVLIN1	IEEVLIN	
	IEEVLNKT		IEEVRJCL	IEEPSN
IEEVLOUT	IEEVLOUT	IEEVLOUT	IEEVLOUT	
IEEVMNT1	IEEVMNT1	IEEVMNT1	IEEVMNT1	
IEEVMNT2	IEEVMNT2	IEEVMNT2	IEEVMNT2	
IEEVOMSG	IEEVOMSG	IEEVOMSG	IEEVOMSG	
IEEVPRES	IEEVPRES	IEEVPRES	IEEVPRES	

(Continued)

CSECT Name	Assembly Module (MODLIB) Name	Entry Point Name	Load Module Name	Alias Name
IEEVRC	IEEVRC	IEEVRC	IEEVRC	
IEEVRCTL	IEEVRCTL	IEEVRCTL	IEEVRCTL	
IEEVRFRX	IEEVRFRX	IEEVRFRX	IEEVLIN IEECVCTI	
IEEVRJCL	IEEVRJCL	IEEVRJCL	IEEVRJCL	IEEPSN
IEEVSIPL	IEEVSIPL	IEEVSIPL	IEEVSIPL	
IEEVSMSG	IEEVSMSG	IEEVSMSG	IEEVICTL IEEVMNT2 IEFIRC	
IEEVSTAR	IEEVSTAR	IEEVSTAR	IEEVSTAR	
IEEVWAIT	IEEVWAIT	IEEVWAIT	IEEVWAIT	IEEVWDAR
IEEXEDNA	IEEXEDNA	IEEXEDNA	IEEXEDNA	
IEE00110	IEE00110	IEE00110	IGC00110	
IEE0003D	IEE0003D	IEE0003D	IGC0003D	
IEE0303D	IEE0303D	IEE0303D	IGC0303D	
IEE0503D	IEE0503D	IEE0503D	IGC0503D IEE0503D	
IEE0603D	IEE0603D	IEE0603D	IGC0603D	
IEE0703D	IEE0703D	IEE0703D	IGC0703D	
IEE0803D	IEE0803D	IEE0803D	IGC0803D	
IEE0903D	IEE0903D	IEE0903D	IGC0903D	
IEE1A03D	IEE1A03D	IEE1A03D	IGC1203D	
IEE1B03D	IEE1B03D	IEE1B03D	IGC1B03D	
IEE10110	IEE10110	IEE10110	IGC10110	
IEE1103D	IEE1103D	IEE1103D	IGC1103D	
IEE11110	IEE11110	IEE11110	IGC11110	
IEE1203D	IEE1203D	IEE1203D	IGC1203D	
IEE12110	IEE12110	IEE12110	IGC12110	
IEE1403D	IEE1403D	IEE1403D	IGC1403D	
IEE1603D	IEE1603D	IEE1603D	IGC1603D	
IEE20110	IEE20110	IEE20110	IGC20110	
IEE2103D	IEE2103D	IEE2103D	IGG2103D	
IEE21110	IEE21110	IEE21110	IGC21110	
IEE22110	IEE22110	IEE22110	IGC22110	
IEE2303D	IEE2303D	IEE2303D	IGC2303D	
IEE23110	IEE23110	IEE23110	IGC23110	
IEE2903D	IEE2903D	IEE2903D	IGC2903D	
IEE30110	IEE30110	IEE30110	IGC30110	
IEE3103D	IEE3103D	IEE3103D	IGC3103D	
IEE31110	IEE31110	IEE31110	IGC31110	
IEE3203D	IEE3203D	IEE3203D	IGC3203D	
IEE32110	IEE32110	IEE32110	IGC32110	
IEE3303D	IEE3303D	IEE3303D	IGC3303D	
IEE3503D	IEE3503D	IEE3503D	IGC3503D	
IEE3603D	IEE3603D	IEE3603D	IGC3603D	
IEE3703D	IEE3703D	IEE3703D	IGC3703D	
IEE3803D	IEE3803D	IEE3803D	IGC3803D	
IEE40110	IEE40110	IEE40110	IGC40110	
IEE4103D	IEE4103D	IEE4103D	IGC4103D	
IEE4203D	IEE4203D	IEE4203D	IGC4203D	
IEE4303D	IEE4303D	IEE4303D	IGC4303D	
IEE4403D	IEE4403D	IEE4403D	IGC4403D	
IEE4503D	IEE4503D	IEE4503D	IGC4503D	
IEE4603D	IEE4603D	IEE4603D	IGC4603D	

(Continued)

CSECT Name	Assembly Module (MODLIB) Name	Entry Point Name	Load Module Name	Alias Name
IEE4703D	IEE4703D	IEE4703D	IGC4703D	
IEE4803D	IEE4803D	IEE4803D	IGC4803D	
IEE4903D	IEE4903D	IEE4903D	IGC4903D	
IEE50110	IEE50110	IEE50110	IGC50110	
IEE5103D	IEE5103D	IEE5103D	IGC5103D	
IEE5203D	IEE5203D	IEE5203D	IGC5203D	
IEE5303D	IEE5303D	IEE5303D	IGC5303D	
IEE5403D	IEE5403D	IEE5403D	IGC5403D	
IEE5503D	IEE5503D	IEE5503D	IGC5503D	
IEE5603D	IEE5603D	IEE5603D	IGC5603D	
IEE5903D	IEE5903D	IEE5903D	IGC5903D	
IEE60110	IEE60110	IEE60110	IGC60110	
IEE6303D	IEE6303D	IEE6303D	IGC6303D	
IEE6403D	IEE6403D	IEE6403D	IGC6403D	
IEE6503D	IEE6503D	IEE6503D	IGC6503D	
IEE6603D	IEE6603D	IEE6603D	IGC6603D	
IEE6703D	IEE6703D	IEE6703D	IGC6703D	
IEE6803D	IEE6803D	IEE6803D	IGC6803D	
IEE6903D	IEE6903D	IEE6903D	IGC6903D	
IEE7103D	IEE7103D	IEE7103D	IGC7103D	
IEE7203D	IEE7203D	IEE7203D	IGC7203D	
IEE7303D	IEE7303D	IEE7303D	IGC7303D	
IEE7503D	IEE7503D	IEE7503D	IGC7503D	
IEE7603D	IEE7603D	IEE7603D	IGC7603D	
IEE7703D	IEE7703D	IEE7703D	IGC7703D	
IEE7803D	IEE7803D	IEE7803D	IGC7803D	
IEE7903D	IEE7903D	IEE7903D	IGC7903D	
IEE8503D	IEE8503D	IEE8503D	IGC8503D	
IEE8603D	IEE8603D	IEE8603D	IGC8603D	
IEFACT	IEFACT	IEFACT	IEFVHA	IEFVHCB
IEFACTLK	IEFSMFLK	IEFACTLK	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
	IEFACTFK	IEFACTLK	IEFW21SD	IEFVM1
				IEFXA
				IEFVMCVL
IEFACTRT	IEFACTRT	IEFACTRT	IEFSD061	IEFSD064
				IEFSD104
				IEFW42SD
			IEFW21SD	IEFVMCVL
				IEFVM1
				IEFXA
IEFBR14	IEFBR14	IEFBR14	IEFBR14	
IEFCNVRT	IEFCNVRT	CNVRT	IEFLOCDQ	LOC
				LOCCAN
				LOCDQ
			IEFVRR1	
IEFCVOL1	IEFCVFAK	IEFCVOL1	IEFW21SD	IEFVMCVL
		IEFCVOL3		IEFVM1
		IEFCVOL2		IEFXA
			IEFVM6LS	IEFXJ000
				IEFXJX5A
				IEFV15XL
				IEFXK000
			IEFWD000	IEFW41SD
			IEFXVAVR	IEFXV001
	IEFMCVOL	IEFCVOL1	IEFMCVOL	IEFCVOL1
		IEFCVOL2		IEFCVOL2
		IEFCVOL3		IEFCVOL3
IEFCVOL2	IEFMCVOL	IEFCVOL1	IEFMCVOL	IEFCVOL1
		IEFCVOL2		IEFCVOL2
		IEFCVOL3		IEFCVOL3
IEFCVOL3	IEFMCVOL	IEFCVOL1	IEFMCVOL	IEFCVOL1

(Continued)

CSECT Name	Assembly Module (MODLIB) Name	Entry Point Name	Load Module Name	Alias Name
		IEFCVOL2		IEFCVOL2
		IEFCVOL3		IEFCVOL3
IEFDEVPT	IEFDEVPT	IEFDEVPT	IEFWA000	IEFWC000
			IEFW21SD	IEFVMCVL
				IEFVM1
				IEFXA
IEFDSSRP	IEFDSDRP	IEFDSDRP	IEFDSDRP	
IEFDSLST	IEFDSLST	IEFDSLST	IEFSD061	IEFSD104
				IEFSD064
				IEFW42SD
IEFDSOAL	IEFSD161	IEFSD061	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
	IEFDSOAL	IEFDSOAL	IEFDSOAL	
IEFDSOCP	IEFDSOCP	IEFDSOCP	IEFDSO	
IEFDSOCR	IEFDSOCR	IEFDSOCR	IEFDSOCR	
IEFDSOFB	IEFDSOFB	IEFDSOFB	IEFDSOFB	
IEFDSOLP	IEFDSOLP	IEFDSOLP	IEFDSOLP	
IEFDSOSM	IEFDSOSM	IEFDSODM	IEFDSOSM	
IEFDSOWR	IEFDSOWR	IEFDSOWR	IEFDSOWR	
IEFDSTBL	IEFDSTBL	IEFDSTBL	IEFSD061	
				IEFSD064
				IEFSD104
				IEFW42SD
IEFDSTRT	IEFDSTRT	IEFDSTRT	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFIDMPM	IEFIDMPM	IEFIDMPM	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFIDUMP	IEFIDUMP	IEFIDUMP	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFIIC	IEFIIC	IEFIIC	IEFIIC	
IEFIRC	IEFIRC	IEFIRC	IEFIRC	
IEFIRCB	IEFIRCB	IEFIRCB	IEFQMSSS	IEFIRCB
IEFK1MSG	IEFK1MSG	IEFK1MSG	IEEVPRES	
IEFLOCDQ	IEFLOCDQ	LOCDQ	IEFLOCDQ	LOCDQ
		LOC		LOC
		LOCCAN		LOCCAN
			IEFVRR1	
IEFORMAT	IEFORMAT	IEFORMAT	IEFSQINT	
IEFPRES	IEFPRES	IEFPRES	IEEVPRES	
IEFQAGST	IEFQAGST	IEFQAGST	IEFVMF	
			IEFVMB	
			IEFQMSSS	IEFIRCB
IEFQASGN	IEFQASGQ	IEFQASGN	IEFQMSSS	IEFIRCB
			IEFDSDRP	
			IEFMCVOL	IEFCVOL3
				IEFCVOL1
				IEFCVOL2
			IEFVMB	
			IEFVMF	
			IEFSD304	
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFSD062	IEFV4221
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFW21SD	IEFVMCVL

(Continued)

CSECT Name	Assembly Module (MODLIB) Name	Entry Point Name	Load Module Name	Alias Name
				IEFXA
				IEFVM1
			IEFVRRC	IEFVRRCA
				IEFVRRCB
			IEFWD000	IEFW41SD
IEFQASNM	IEFQASGQ	IEFQASGN	IEFQMSSS	IEFIRCB
			IEFDSDRP	
			IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
			IEFVMB	
			IEFVMF	
			IEFSD304	
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFSD062	IEFV4221
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFW21SD	IEFVMCVL
				IEFVM1
				IEFXA
			IEFVRRC	IEFVRRCA
				IEFVRRCB
			IEFWD000	IEFW41SD
IEFQDELE	IEFQDELQ	IEFQDELE	IEFQDELE	
			IEFVMD	
			IEFVMF	
			IEFSD304	
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFSD080	IEFSD079
				IEEVWTR1
			IEFVRR3	IEFVR3AE
			IEESD576	IEESD580
IEFQINTZ	IEFQINTZ	IEFQINTZ	IEFQINTZ	
IEFQMDQ2	IEFQMDQQ	IEFQMDQ2	IEFQMDQQ	IEFQMDQ2
			IEFSD080	IEFSD079
				IEEVWTR1
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFVMF	
IEFQMNQ2	IEFQMNQQ	IEFQMNQ2	IEFVMF	
			IEFVMB	
			IEFSD061	IEFW42SD
				IEFSD065
				IEFSD104
			IEFSD304	
			IEFQMSSS	IEFIRCB
			IEFQMNQ2	
			IEFDSOCR	
			IEESD577	
			IEESD579	
IEFQMRAW	IEFQMRAW	IEFQMRAW	IEFDSOCR	
			IEFDSO	
			IEFDSDRP	
			IEFQMRAW	
			IEFQMSSS	IEFIRCB
			IEFMCVOL	IEFCVOL3
				IEFCVOL1

(Continued)

CSECT Name	Assembly Module (MODLIB) Name	Entry Point Name	Load Module Name	Alias Name
				IEFCVOL2
			IEFSD304	
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFVMF	
			IEFSD062	IEFV4221
			IEFSD086	IEF086SD
			IEFSD085	IEFSD071
				IEF085SD
				IEF850SD
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
			IEFVRRC	IEFVRRCB
				IEFVRRCA
			IEFWD000	IEFW41SD
			IEFXVAVR	IEFXV001
IEFQMSGV	IEFQMSGV		IEFSQINT	
IEFQMSSS	IEFQMDUM	IEFQMSSS	IEFVHA	IEFVHCB
	IEFQBVMS	IEFQMSSS	IEFVMF	
			IEFQMSSS	IEFIRCB
			IEFDSDRP	
	IEFQMLK1	IEFQMSSS	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFXVAVR	IEFXV001
			IEFWD000	IEFW41SD
			IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFMCVOL	IEFCVOL3
				IEFCVOL1
				IEFCVOL2
IEFQMUNC	IEFQMUNQ	IEFQMUNC	IEFQMUNC	
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFVMF	
			IEFSD080	IEFSD079
				IEEVWTR1
IEFQMWTO	IEFQMRAW	IEFQMRAW	IEFDSOCP	
			IEFDSO	
			IEFDSDRP	
			IEFQMRAW	
			IEFQMSSS	IEFIRCB
			IEFMCVOL	IEFCVOL3
				IEFCVOL1
				IEFCVOL2
			IEFSD304	
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFVMF	
			IEFSD062	IEFV4221
			IEFSD086	IEF086SD

(Continued)

CSECT Name	Assembly Module (MODLIB) Name	Entry Point Name	Load Module Name	Alias Name
			IEFSD085	IEFSD071
				IEF085SD
				IEF850SD
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
			IEFVRRC	IEFVRRCB
				IEFVRRCA
			IEFWD000	IEFW41SD
			IEFXVAVR	IEFXV001
IEFRCLN1	IEFRCLN1	IEFRCLN1	IEFRCLN1	
IEFRCLN2	IEFRCLN2	IEFRCLN2	IEFRCLN2	
IEFRDWRT	IEFRDWRT	WRT	IEFLOCDQ	LOC
		RD		LOCCAN
				LOCDQ
			IEFVRR1	
IEFRPREP	IEFRPREP	IEFRPREP	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFRSTRT	IEFRSTRT	IEFSMR	IEFRSTRT	IEFSMR
		IEFRSTRT		
IEFSCAN	IEFSCAN	IEFSCAN	IEFWA000	IEFWC000
		UCBSCAN	IEFW21SD	IEFVMCVL
		UCBCONT		IEFVM1
				IEFXA
IEFSDPPT	IEFSDPPT		IEFSD061	IEFSD104
				IEFW42SD
				IEFV4221
				IEFSD064
IEFSDTTE	IEFSDTTE	IEFSDTTE	IEFSDTTE	
IEFSDXXX	IEFSDXXX	IEFSDXXX	IEFSDXXX	
IEFSDXYZ	IEFSDXYZ	IEFSDXYZ	IEFSDXYZ	
IEFSD0XX	IEFSD0XX	IEFSD0XX	IEFSD0XX	
IEFSD012	IEFVSD12	IEFSD012	IEFVHA	IEFVHCB
IEFSD055	IEFSD055	IEFSD055	IEFSQINT	
IEFSD060	IEFSD160	IEFSD060	IEFSD060	
	IEF060SD	IEFSD060	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFSD061	IEF061SD	IEFSD061	IEFSD062	IEFV4221
			IEFSD060	
	IEFSD161	IEFSD061	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFSD062	IEFSD162	IEFSD062	IEFSD062	IEFV4221
IEFSD063	IEFSD103	IEFSD063	IEFSD062	IEFV4221
IEFSD064	IEFSD164	IEFSD064	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
	IEF064SD	IEFSD064	IEFSD062	
IEFSD066	IEFSD166	IEFSD066	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFSD068	IEFSD168	IEFSD068	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFDSOCR	
IEFSD070	IEFSD070	IEFSD070	IEFSD070	
IEFSD071	IEFSD171	IEFSD071	IEFSD085	IEF850SD
				IEF085SD

(Continued)

CSECT Name	Assembly Module (MODLIB) Name	Entry Point Name	Load Module Name	Alias Name
				IEFSD071
IEFSD078	IEF078SD	IEFSD078	IEFSD080	IEFSD079
				IEEVWTR1
	IEFSD078	IEFSD078	IEFSD078	
IEFSD079	IEF079SD	IEFSD079	IEFSD078	
	IEFSD079	IEFSD079	IEFSD080	IEFSD079
				IEEVWTR1
IEFSD080	IEFSD080	IEFSD080	IEFSD080	IEFSD079
				IEEVWTR1
IEFSD081	IEFSD081	IEFSD081	IEFSD080	IEFSD079
				IEEVWTR1
IEFSD082	IEFSD082	IEFSD082	IEFSD080	IEFSD079
				IEEVWTR1
IEFSD083	IEFSD083	IEFSD083	IEFSD080	IEFSD079
				IEEVWTR1
IEFSD084	IEFSD084	IEFSD084	IEFSD080	IEFSD079
				IEEVWTR1
IEFSD085	IEFSD085	IEFSD085	IEFSD085	IEF085SD
		IEF085SD		IEFSD071
		IEF850SD		IEF850SD
IEFSD086	IEFSD086	IEFSD086	IEFSD086	IEF086SD
		IEF086SD		
IEFSD087	IEFSD087	IEFSD087	IEFSD087	
IEFSD088	IEFSD088	IEFSD088	IEFSD087	
			IEFSD094	
			IEFSD086	IEF086SD
IEFSD089	IEFSD089	IEFSD089	IEFSD086	IEF086SD
			IEFSD094	
			IEFSD087	
IEFSD090	IEFVSD13	IEFSD090	IEFVHA	IEFVHCB
IEFSD094	IEFSD094	IEFSD094	IEFSD094	
IEFSD095	IEFSD095	IEFSD095	IEFSD094	
	IEFSD195	IEFVAWAT	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFWD000	IEFW41SD
IEFSD096	IEFSD096	IEFSD096	IEFWD000	IEFW41SD
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
IEFSD097	IEFSD097	IEFSD097	IEFWD000	IEFW41SD
				IEFSD104
				IEFSD064
IEFSD101	IEFSD101	IEFSD101	IEFSD061	IEFW42SD
IEFSD102	IEFSD102	IEFSD102	IEFSD102	
IEFSD104	IEFSD104	IEFSD104	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFSD105	IEFSD105	IEFSD105	IEFSD105	
IEFSD110	IEFSD110	IEFSD110	IEFSD062	IEFV4221
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFVMF	
IEFSD111	IEFSD111	IEFSD111	IEFVMF	
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFSD062	IEFV4221
			IEFWD000	IEFWC000
			IEFVM6LS	IEFXK000
				IEFXJ000

(Continued)

CSECT Name	Assembly Module (MODLIB) Name	Entry Point Name	Load Module Name	Alias Name
				IEFXJX5A
				IEFV15XL
			IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
			IEFXVAVR	IEFXV001
			IEFDSOCR	
			IEFMCVOL	IEFCVOL3
			IEFCVOL1	IEFCVOL2
				IEFCVOL1
IEFSD112	IEFSD112	IEFSD112	IEFDSOCR	
			IEFVRR3	IEFVR3AE
			IEFSD062	IEFV4221
			IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFVMF	
IEFSD180	IEFSD180	IEFSD18	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFSD263	IEFSD263	IEFSD263	IEFSD263	
IEFSD300	IEFSD300	IEFSD300	IEFSD300	
	IEF300SD	IEFSD300	IEFSQINT	
IEFSD301	IEFSD301	IEFSD301	IEFSD300	
IEFSD302	IEFSD302	IEFSD302	IEFSD300	
IEFSD303	IEFSD303	IEFSD303	IEFSD300	
IEFSD304	IEFSD304	IEFSD304	IEFSD304	
	IEF304SD	IEFSD304	IEFSQINT	
IEFSD305	IEFSD305	IEFSD305	IEFSD304	
IEFSD308	IEFSD308	IEFSD308	IEFSD308	
IEFSD310	IEFSD310	IEFSD310	IEFSD304	
			IEFSD300	
IEFSD311	IEFSD311	SD55MSG3	IEFSQINT	
		SD55MSG2		
		SD55MSG1		
		IEFSD311		
IEFSD312	IEFSD312	SD304MG2	IEFSD304	
		SD304MG1	IEFSD300	
		IEFSD312		
		SD305MG1		
IEFSD514	IEFSD514	IEFSD514	IEFSD304	
			IEFSD062	IEFV4221
			IEFVRRC	IEFVRRCB
				IEFVRRCA
IEFSD572	IEFYRCDS	IEFYRCDS	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFSD60M	IEFSD160	IEFSD060	IEFSD060	
IEFSD71M	IEFSD171	IEFSD071	IEFSD085	IEF850SD
				IEF085SD
				IEFSD071
IEFSD83M	IEFSD083	IEFSD083	IEFSD080	IEFSD079
				IEEVWTR1
IEFSD85M	IEFSD085	IEF850SD	IEFSD085	IEF850SD
		IEF085SD		IEF085SD
		IEFSD085		IEFSD071
IEFSD86M	IEFSD086	IEF086SD	IEFSD086	IEF086SD
		IEFSD086		
IEFSD87M	IEFSD087	IEFSD087	IEFSD087	
IEFSD89M	IEFSD089	IEFSD089	IEFSD094	
			IEFSD087	
			IEFSD086	IEF086SD
IEFSGOPT	IEFSGOPT	IEFSGOPT	IEFWA000	IEFWC000
IEFSMFAT	IEFSMFAT	IEFSMFAT	IEFSD263	

(Continued)

CSECT Name	Assembly Module (MODLIB) Name	Entry Point Name	Load Module Name	Alias Name
IEFSMFIE	IEFSMFIE	IEFSMFIE	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFSMFWI	IEFSMFWI	IEFSMFWI	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFUJI	IEFUJI	IEFUJI	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFUJV	IEFUJV	IEFUJV	IEFUJV	
IEFUSI	IEFUSI	IEFUSI	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFUSO	IEFUSO	IEFUSO	IEFSD263	
IEFUTL	IEFUTL	IEFUTL	IEFSD263	
IEFVDA	IEFVDA	IEFVDA	IEFVHA	IEFVHCB
IEFVDBSD	IEFVDBSD	IEFVDBSD	IEFVHA	IEFVHCB
IEFVEA	IEFVEA	IEFVEA	IEFVHA	IEFVHCB
IEFVFA	IEFVFA	IEFVFA	IEFVHA	IEFVHCB
IEFVFB	IEFVFB	IEFVFB	IEFVHA	IEFVHCB
IEFVGI	IEFVGI	IEFVGI	IEFVHA	IEFVHCB
IEFVGK	IEFVGK	IEFVGK	IEFVHA	IEFVHCB
IEFVGM	IEFVGM	IEFVGM	IEFVHA	IEFVHCB
			IEFVINA	
IEFVGM1	IEFVGM1		IEFVGM1	
IEFVGM10	IEFVGM10		IEFVGM10	
IEFVGM11	IEFVGM11		IEFVGM11	
IEFVGM12	IEFVGM12		IEFVGM12	
IEFVGM13	IEFVGM13		IEFVGM13	
IEFVGM14	IEFVGM14		IEFVGM14	
IEFVGM15	IEFVGM15		IEFVGM15	
IEFVGM16	IEFVGM16		IEFVGM16	
IEFVGM17	IEFVGM17		IEFVGM17	
IEFVGM18	IEFVGM18		IEFVGM18	
IEFVGM19	IEFVGM19		IEFVGM19	
IEFVGM2	IEFVGM2		IEFVGM2	
IEFVGM3	IEFVGM3		IEFVGM3	
IEFVGM4	IEFVGM4		IEFVGM4	
IEFVGM5	IEFVGM5		IEFVGM5	
IEFVGM6	IEFVGM6		IEFVGM6	
IEFVGM7	IEFVGM7		IEFVGM7	
IEFVGM70	IEFVGM70		IEFVGM70	
IEFVGM71	IEFVGM71		IEFVGM71	
IEFVGM78	IEFVGM78		IEFVGM78	
IEFVGM8	IEFVGM8		IEFVGM8	
IEFVGM9	IEFVGM9		IEFVGM9	
IEFVGS	IEFVGS	IEFVGS	IEFVHA	IEFVHCB
IEFVGT	IEFVGT	IEFVGT	IEFVHA	IEFVHCB
IEFVHA	IEFVHA	IEFVHA	IEFVHA	IEFVHCB
IEFVHAA	IEFVHAA	IEFVHAA	IEFVHA	IEFVHCB
IEFVHB	IEFVHB	IEFVHB	IEFVHA	IEFVHCB
IEFVHC	IEFVHC	IEFVHC	IEFVHA	IEFVHCB
IEFVHCB	IEFVHCB	IEFVHCB	IEFVHA	IEFVHCB
IEFVHE	IEFVHE	IEFVHE	IEFVHA	IEFVHCB
IEFVHEB	IEFVHEB	IEFVHEB	IEFVHA	IEFVHCB
IEFVHEC	IEFVHEC	IEFVHEC	IEFVHA	IEFVHCB
IEFVHF	IEFVHF	IEFVHF	IEFVHA	IEFVHCB
IEFVHG	IEFVHG	IEFVHG	IEFVHA	IEFVHCB
IEFVHH	IEFVHH	IEFVHH	IEFVHA	IEFVHCB
IEFVHHB	IEFVHHB	IEFVHHB	IEFVHA	IEFVHCB
IEFVHL	IEFVHL	IEFVHL	IEFVHA	IEFVHCB
IEFVHM	IEFVHM	IEFVHM	IEFVHA	IEFVHCB
IEFVHN	IEFVHN	IEFVHN	IEFVHN	
IEFVHQ	IEFVHQ	IEFVHQ	IEFVHA	IEFVHCB

(Continued)

CSECT Name	Assembly Module (MODLIB) Name	Entry Point Name	Load Module Name	Alias Name
			IEFVINA	
IEFVHR	IEFVHR	IEFVHR	IEFVHA	IEFVHCB
			IEFVINA	
IEFVH1	IEFVH1	IEFVH1	IEFVH1	
IEFVH2	IEFVH2	IEFVH2	IEFVH1	
IEFVINA	IEFVINA	IEFVINA	IEFVINA	
IEFVINB	IEFVINB	IEFVINB	IEFVINA	
			IEFVINB	
IEFVINC	IEFVINC	IEFVINC	IEFVINA	
IEFVIND	IEFVIND	IEFVIND	IEFVHA	IEFVHCB
IEFVINE	IEFVINE	IEFVINE	IEFVHA	
IEFVJ	IEFVJIMP	IEFVJ	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFVJA	IEFVJA	IEFVJA	IEFVHA	IEFVHCB
IEFVJMSG	IEFVJMSG	IEFVJMSG	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFVK	IEFVKIMP	IEFVK	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFVKG	IEFVKG	IEFVKG	IEFVHA	IEFVHCB
IEFVKMSG	IEFVKMSG	IEFVKMJ1	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFVMA	IEFVMA	IEFVMA	IEFVMA	
IEFVMB	IEFVMB	IEFVMB	IEFVMB	
IEFVMC	IEFVMC	IEFVMC	IEFVMC	
IEFVMCVL	IEFVMFAK	IEFVMCVL	IEFMCVOL	IEFCVOL3
				IEFCVOL1
				IEFCVOL2
IEFVMD	IEFVMD	IEFVMD	IEFVMD	
IEFVME	IEFVME	IEFVME	IEFVME	
IEFVMF	IEFVMF	IEFVMF	IEFVMF	
IEFVMG	IEFVMG	IEFVMG	IEFVMF	
IEFVMH	IEFVMH	IEFVMH	IEFVMF	
IEFVM1	IEFVMLS1	IEFVMCVL	IEFW21SD	IEFVMCVL
		IEFVM		IEFVM1
		VM7750		IEFXA
		VM7742		
		VM7700		
		VM7370		
		VM7130		
		VM7090		
		VM7070		
		VM7060		
		VM7055AA		
		VM7055		
		VM7000		
		IEFVMQMI		
		VM7950		
		VM7900		
		VM7850		
	IEFVMMS1	IEFVM1	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFWD000	IEFWC000
			IEFMCVOL	IEFCVOL3
				IEFCVOL1
				IEFCVOL2
IEFVM2	IEFVM2LS	VM7100	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1

(Continued)

CSECT Name	Assembly Module (MODLIB) Name	Entry Point Name	Load Module Name	Alias Name
IEFVM3	IEFVM3LS	VM7150	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFVM4	IEFVM4LS	VM7200	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFVM5	IEFVM5IS	VM7300	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFVM6	IEFVMLK5	IEFVMLK5	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
	IEFVMLS6	IEFVMSGR	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFMCVOL	IEFCVOL3
				IEFCVOL1
				IEFCVOL2
IEFVM7	IEFVMLS7	IEFVM7	IEFMCVOL	IEFCVOL3
				IEFCVOL1
				IEFCVOL2
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
IEFVM76	IEFVM76	VM7600	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFVRRC	IEFVRRC	IEFVRRCB	IEFVRRC	IEFVRRCB
		IEFVRRCA		IEFVRRCA
		IEFVRRC		
IEFVRR1	IEFVRR1	IEFVRR1	IEFVRR1	
IEFVRR2	IEFVRR2	IEFV2AE	IEFVRR2	IEFVR2AE
		IEFVRR2		
IEFVRR3	IEFVRR3	IEFV3AE	IEFVRR3	IEFVR3AE
		IEFVRR3		
IEFVSDRA	IEFVSDRA	IEFVSDRA	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
			IEFSD304	
IEFVSDRD	IEFVSDRD	IEFVSDRD	IEFSD304	
IEFV15XL	IEFSD551	IEFSD551	IEFWA000	IEFWC000
			IEFXVAVR	IEFXV001
IEFWAD	IEFWAD	IEFWAD	IEFW21SD	IEFVMCVL
				IEFVM1
				IEFXA
IEFWA000	IEFWAFAK		IEFW21SD	IEFVMCVL
				IEFVM1
				IEFXA
IEFWA002	IEFWA000	IEFUCBL	IEFWA000	IEFWC000
IEFWA7	IEFWA000	IEFUCBL	IEFWA000	IEFWC000
IEFWC000	IEFWCFAK	IEFWC000	IEFWA000	IEFWC000
			IEFXVAVR	IEFXV001
	IEFWCIMP	IEFWC000	IEFX5000	IEFWD000
				IEFWC000
				IEFW41SD
IEFWD000	IEFWD000	IEFWD000	IEFWD000	IEFW41SD
			IEFXVAVR	IEFXV001
	IEFWDFAK	IEFWD000	IEFWA000	IEFWC000
IEFWD001	IEFWD001	IEFWD001	IEFXVAVR	IEFXV001
			IEFWD000	IEFW41SD
IEFWEXTA	IEFWEXTA	IEFWEXTA	IEFWD000	IEFW41SD

(Continued)

CSECT Name	Assembly Module (MODLIB) Name	Entry Point Name	Load Module Name	Alias Name
				IEFXA
IEFWSTRT	IEFSTRT	IEFSTRT	IEFW21SD	IEFVMCVL
				IEFVM1
IEFWSWIT	IEFWSWIN	IEFWSWIT	IEFWA000	IEFWC000
IEFWTERM	IEFWTERM	IEFWTERM	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFW21SD	IEFSD21Q	IEFW21SD	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFW22SD	IEFSD22Q	IEFW22SD	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFW31SD	IEFSD31Q	IEFW31SD	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFW41SD	IEFSD41Q	IEFW2FAK	IEFSD061	IEFW42SD
		IEFW1FAK		
		IEFW41SD		IEFSD064
				IEFSD104
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFWD000	IEFWC000
	IEF41FAK	IEFW2FAK	IEFW21SD	IEFVMCVL
		IEFW1FAK		IEFVM1
		IEFW41SD		IEFXA
			IEFMCVOL	IEFCVOL3
				IEFCVOL2
				IEFCVOL1
			IEFWA000	IEFWC000
IEFXA	IEFXCSSS	IEFXA	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
	IEFXAFAK	IEFXAFAK	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
IEFXAMSG	IEFXAMSG	IEFXAMSG	IEFW21SD	IEFVMCVL
				IEFXJ000
				IEFXJX5A
				IEFV15XL
IEFXDPTH	IEFXDPTH	IEFXDPTH	IEFWA000	IEFWC000
IEFXH000	IEFXH000	IEFXH000	IEFWA000	IEFWC000
			IEFXVAVR	IEFXV001
IEFXJMSG	IEFXJMSG	MSOFF	IEFVM6LS	IEFXK000
		MSSYS		IEFXJ000
		MSRCV		IEFXJX5A
				IEFV15XL
IEFXJX5A	IEFSD552	IEFSD552	IEFXVAVR	IEFXV001
			IEFWA000	IEFWC000
IEFXJ000	IEFXJFAK	IEFXJ000	IEFWA000	IEFWC000
			IEFXVAVR	IEFXV001
			IEFWD000	IEFW41SD
	IEFXJIMP	IEFXJX5A	IEFVM6LS	IEFXK000
		IEFV15XL		IEFXJ000
		IEFXJ000		IEFXJX5A
				IEFV15XL
IEFXKMSG	IEFXKMSG	IEFXKMSG	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A

(Continued)

CSECT Name	Assembly Module (MODLIB) Name	Entry Point Name	Load Module Name	Alias Name
				IEFV15XL
IEFXK000	IEFXKIMP	IEFXK000	IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
	IEFXKFAK	IEFXKFAK	IEFWD000	IEFW41SD
IEFXTDMY	IEFXTDMY	IEFXTDMY	IEFWD000	IEFW41SD
IEFXTMSG	IEFXTMSG	RESMSG	IEFWD000	IEFW41SD
		MESGDDEV		
		PRIAMSG		
		JOBCAN		
		WTORREP		
		RESUNIT		
IEFXT000	IEFXT00D	WDEXIT	IEFWD000	IEFW41SD
IEFXT002	IEFXT002	IEFXT002	IEFWD000	IEFW41SD
		XTTEB3		
		XTTRDJ		
IEFXT003	IEFXT003	XUUB00	IEFWD000	IEFW41SD
IEFXVMSG	IEFXVMSG	IEFXVMSG	IEFXVAVR	IEFXV001
IEFXVNSL	IEFXVNSL	IEFXVNSL	IEFXVAVR	IEFXV001
IEFXV001	IEFXV001	IEFXV001	IEFXVAVR	IEFXV001
	IEFAVFAK		IEFWA000	IEFWC000
IEFXV002	IEFXV002	IEFXV002	IEFXVAVR	IEFXV001
IEFXV003	IEFXV003	IEFXV003	IEFXVAVR	IEFXV001
IEFX3000	IEFX300A	X33B42	IEFXVAVR	IEFXV001
		IEFX3000	IEFWA000	IEFWC000
IEFX5000	IEFX5000	X55C86	IEFWA000	IEFWC000
		X55D3G	IEFXVAVR	IEFXV001
	IEFX5FAK		IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
IEFYN	IEFYNIMP	IEFYN	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFYNMSG	IEFYNMSG	IEFYNMSG	IEFSD061	IEFW42SD
		STRMSG01		
				IEFSD064
				IEFSD104
IEFYP	IEFYPJB3	IEFYP	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFYPMSG	IEFYPMSG	YPPMSG1	IEFSD061	IEFW42SD
		IEFYPMSG		
				IEFSD064
				IEFSD104
IEFYRCDS	IEFYRCDS	IEFYRCDS	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
IEFYS	IEFYSVMS	IEFYS	IEFW21SD	IEFVMCVL
				IEFXA
				IEFVM1
			IEFWD000	IEFW41SD
			IEFVM6LS	IEFXK000
				IEFXJ000
				IEFXJX5A
				IEFV15XL
			IEFMCVOL	IEFCVOL3
				IEFCVOL1
				IEFCVOL2
IEFZG	IEFZGST1	IEFZG	IEFSD061	IEFW42SD
		ZG0J8		
		ZG0J5		IEFSD064

(Continued)

CSECT Name	Assembly Module (MODLIB) Name	Entry Point Name	Load Module Name	Alias Name
				IEFSD104
IEFZGJ	IEFZGJBL	ZP0GM	IEFSD061	
IEFZG2	IEFZGST2	ZGOK09	IEFSD061	IEFW42SD
		IEFZG2		
		ZPOC10		IEFSD064
		ZOOE10		IEFSD104
		ZOOA1		
		ZPOQMGR1		
IEFZH	IEFZHMSG	XPS631	IEFSD061	IEFW42SD
		ZKOE1		
		IEFZH		IEFSD064
		ZG0E60		IEFSD104
		ZK0D1		
IEXAB00	IEFXCSSS	IEFXA	IEFW21SD	IEFVMCVL
				IEFSD104
				IEFXA
				IEFVM1
IEZDCODE	IEZDCODE	IEZDCODE	IEFVMF	
			IEFVINA	
IEZNCODE	IEZNCODE	IEZNCODE	IEFVMB	
			IEFVINA	
IGC0005B	IEFVSMBR	IGC0005B	IGC0005B	
IGC0083	IEESMF8C	IEESMF8C	IGC0008C	
IGC0203E	IEFWTP00	IGC0203E	IGC0203E	
IGC0208E	IEESMFAL	IGC0208E	IGC0208C	
IGC0303E	IEFWTP01	IGC0303E	IGC0303E	
IGC0403E	IEFWTP02	IGC0403E	IGC0403E	
IGC6103D	IGC6103D	IGC6103D	IGC6103D	
IGC6203D	IGC6203D	IGC6203D	IGC6203D	
IGF08501	IGF08501	IGF08501	IGF08501	
IGF08502	IGF08502	IGF08521	IGF08502	
IGF2603D	IGF2603D	IGF2603D	IGF2603D	
IGF29601	IGF29601	IGF29601	IGF29601	
IGF29701	IGF29701	IGF29701	IGF29701	
IGF55301	IGF55301	IGF55301	IGF55301	
IKJNULL	IKJNULL	IKJNULL	IGC5803D	
IKJ5803D	IKJ5803D	IKJ5803D	IGC5803D	
QMTMSD	IEFSD161	IEFSD061	IEFSD061	IEFW42SD
				IEFSD064
				IEFSD104
SPRINTER	IEFPRTXX	SPRINTER	IEFPRINT	IEFPRT

Appendix D: Job Management Charts

A number in the upper right corner of a box (unless otherwise indicated) refers to another chart in this appendix.

Chart 01. Job Management Data Flow

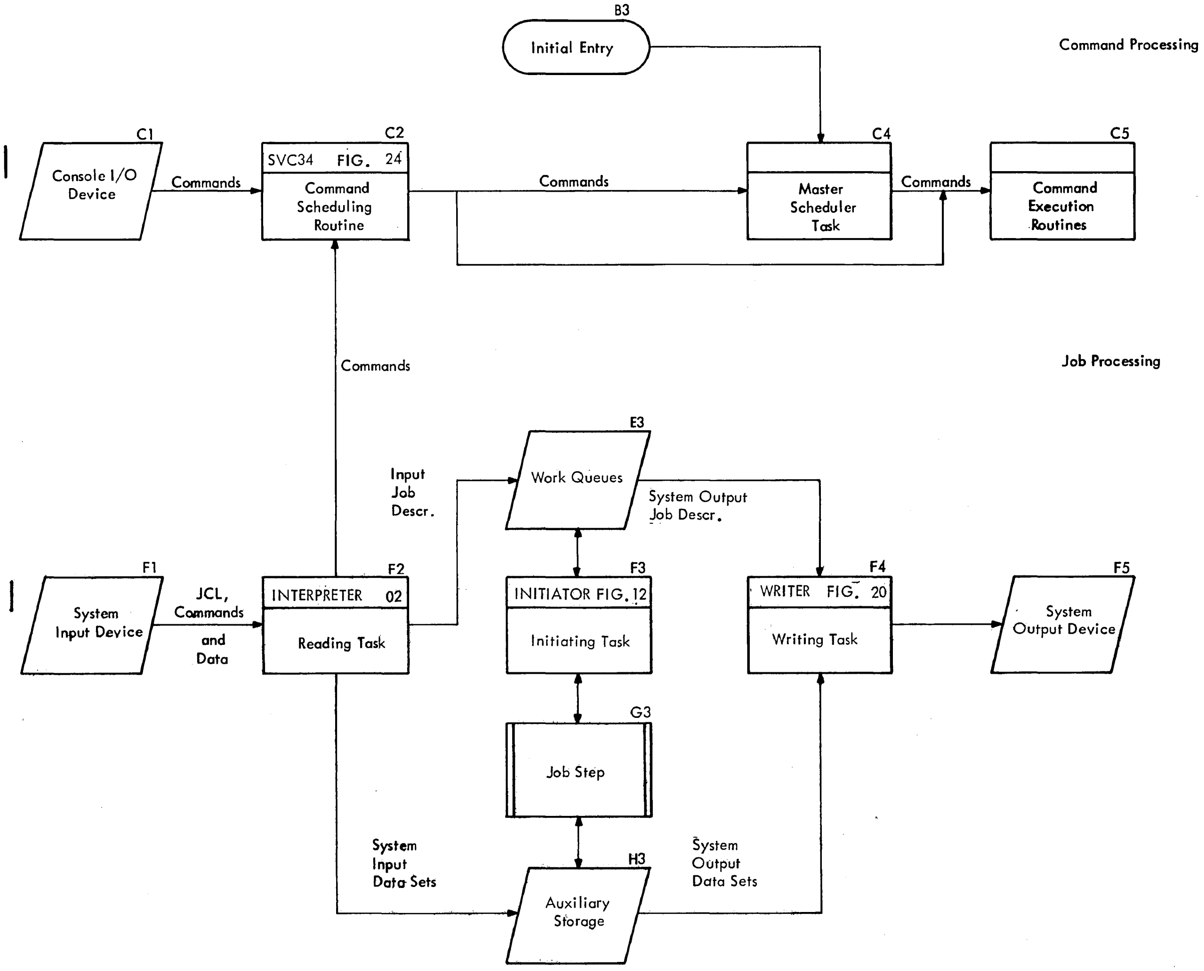


Chart 02. Interpreter Control Flow

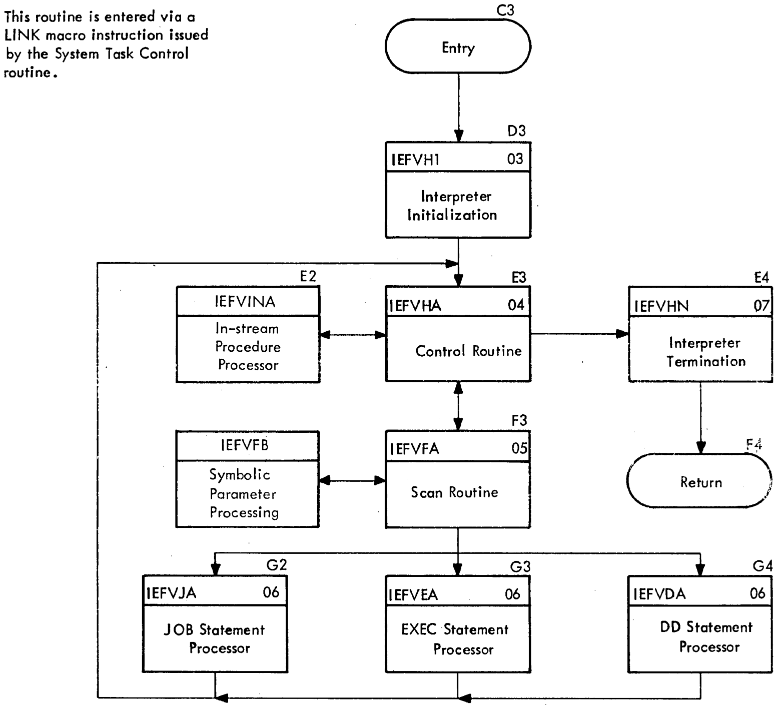


Chart 03. Interpreter Initialization Routine

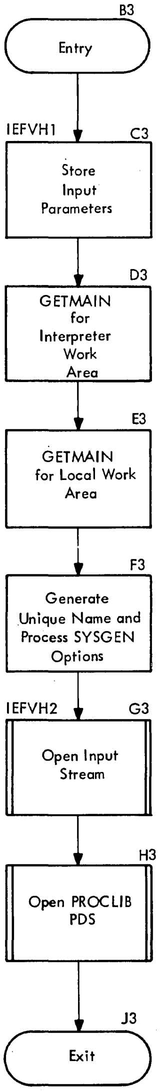

B3
Entry
IEFVH1
C3
Store Input Parameters
D3
GETMAIN for Interpreter Work Area
E3
GETMAIN for Local Work Area
F3
Generate Unique Name and Process SYSGEN Options
IEFVH2
G3
Open Input Stream
H3
Open PROCLIB PDS
J3
Exit

Chart 04. Interpreter Control Routine

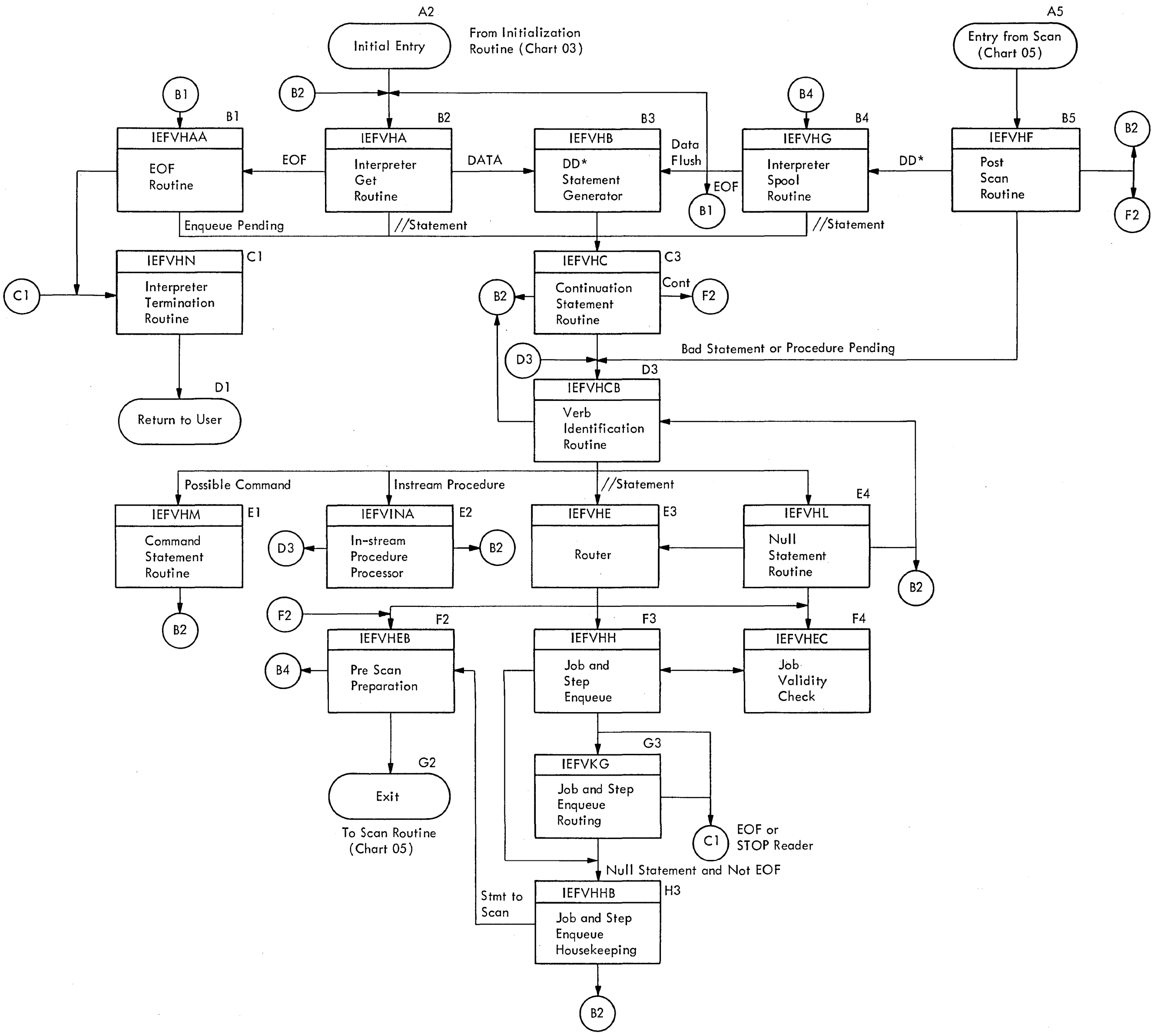
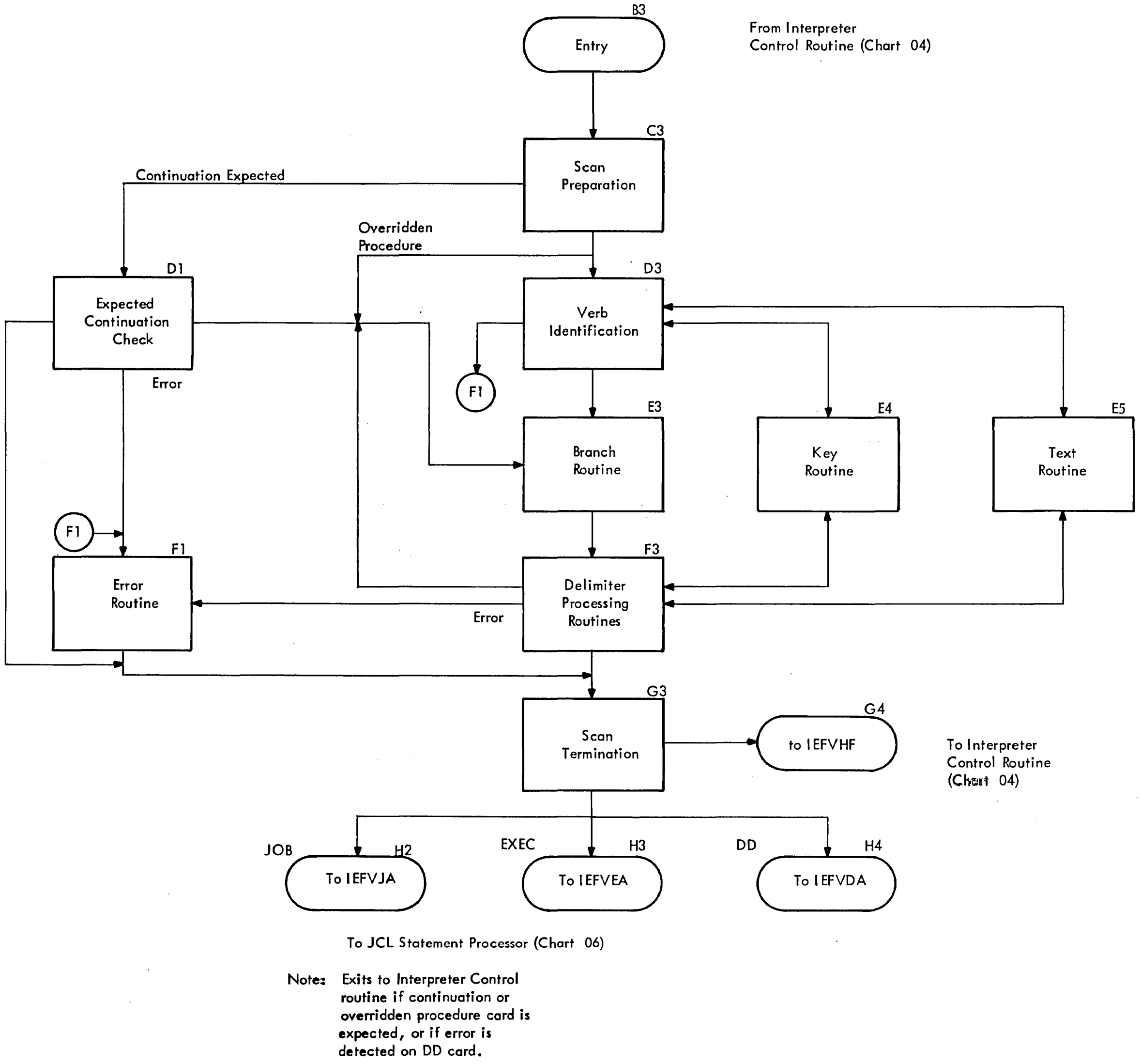

A2
Initial Entry
From Initialization Routine (Chart 03)
A5
Entry from Scan (Chart 05)
B1
B2
B4
B1 IEFVHAA EOF Routine
B2 IEFVHA Interpreter Get Routine
B3 IEFVHB DD* Statement Generator
B4 IEFVHG Interpreter Spool Routine
B5 IEFVHF Post Scan Routine
EOF
DATA
Data Flush
EOF
DD*
B2
F2
Enqueue Pending
//Statement
//Statement
C1 IEFVHN Interpreter Termination Routine
C1
C3 IEFVHC Continuation Statement Routine
Cont
F2
B2
D3
Bad Statement or Procedure Pending
D1 Return to User
D3 IEFVHCB Verb Identification Routine
Possible Command
Instream Procedure
//Statement
E1 IEFVHM Command Statement Routine
E2 IEFVINA In-stream Procedure Processor
E3 IEFVHE Router
E4 IEFVHL Null Statement Routine
D3
B2
B2
B2
F2
F2 IEFVHEB Pre Scan Preparation
F3 IEFVHH Job and Step Enqueue
F4 IEFVHEC Job Validity Check
B4
G2 Exit
To Scan Routine (Chart 05)
G3 IEFVKG Job and Step Enqueue Routing
C1
EOF or STOP Reader
Null Statement and Not EOF
H3 IEFVHHB Job and Step Enqueue Housekeeping
Stmt to Scan
B2

Chart 05. Interpreter Scan Routine



B3
Entry
From Interpreter Control Routine (Chart 04)
C3
Scan Preparation
Continuation Expected
Overridden Procedure
D1
Expected Continuation Check
Error
D3
Verb Identification
F1
E3
Branch Routine
E4
Key Routine
E5
Text Routine
F1
F1
Error Routine
F3
Delimiter Processing Routines
Error
G3
Scan Termination
G4
to IEFVHF
To Interpreter Control Routine (Chart 04)
JOB
H2
To IEFVJA
EXEC
H3
To IEFVEA
DD
H4
To IEFVDA
To JCL Statement Processor (Chart 06)
Note: Exits to Interpreter Control routine if continuation or overridden procedure card is expected, or if error is detected on DD card.

Chart 06. JCL Statement Processors

Note: This chart shows control flow in all five statement processors. Each statement processor is entered from the Interpreter Scan routine (Chart 05) and passes control to the Control routine (Chart 04).

Each statement processor includes several keyword processors, most of which use IEFVGT and IEFVGK as subroutines.

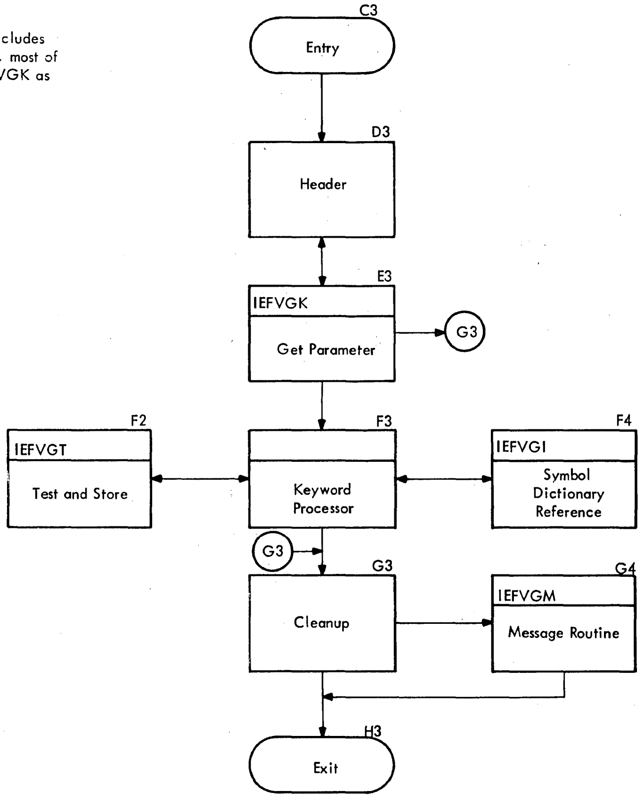


Chart 07. Interpreter Termination Routine

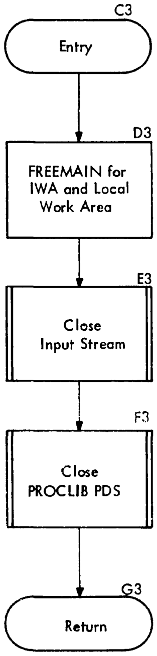
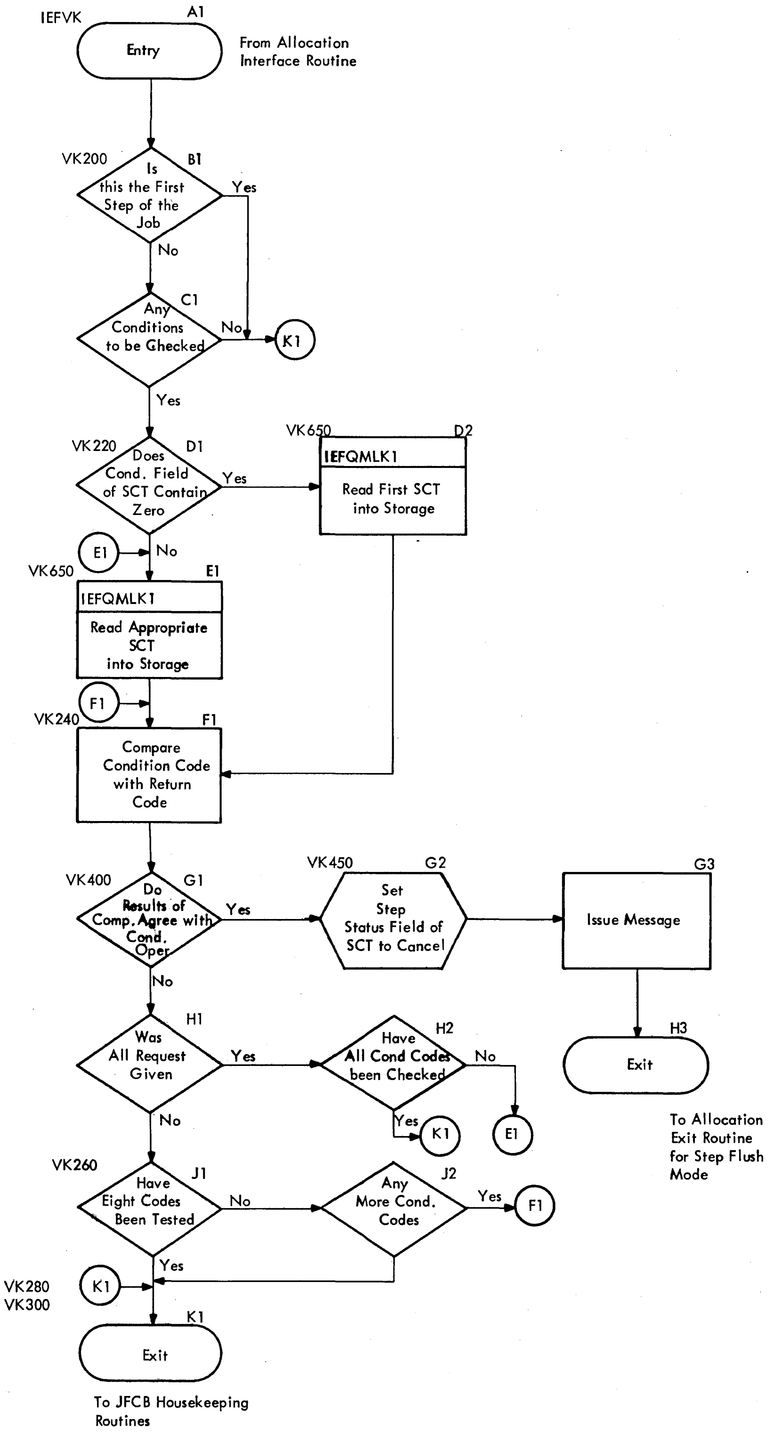


Chart 08. Execute Statement Condition Code Processing Routine



IEFVK
A1
Entry
From Allocation Interface Routine
VK200
B1
Is this the First Step of the Job
Yes
No
C1
Any Conditions to be Checked
No
K1
Yes
VK220
D1
Does Cond. Field of SCT Contain Zero
Yes
VK650
D2
IEFQMLK1
Read First SCT into Storage
E1
No
VK650
E1
IEFQMLK1
Read Appropriate SCT into Storage
F1
VK240
F1
Compare Condition Code with Return Code
VK400
G1
Do Results of Comp. Agree with Cond. Oper
Yes
VK450
G2
Set Step Status Field of SCT to Cancel
G3
Issue Message
No
H1
Was All Request Given
Yes
H2
Have All Cond Codes been Checked
No
Yes
K1
E1
H3
Exit
To Allocation Exit Routine for Step Flush Mode
No
VK260
J1
Have Eight Codes Been Tested
No
J2
Any More Cond. Codes
Yes
F1
Yes
VK280
VK300
K1
K1
Exit
To JFCB Housekeeping Routines

Chart 09. JFCB Housekeeping Routines

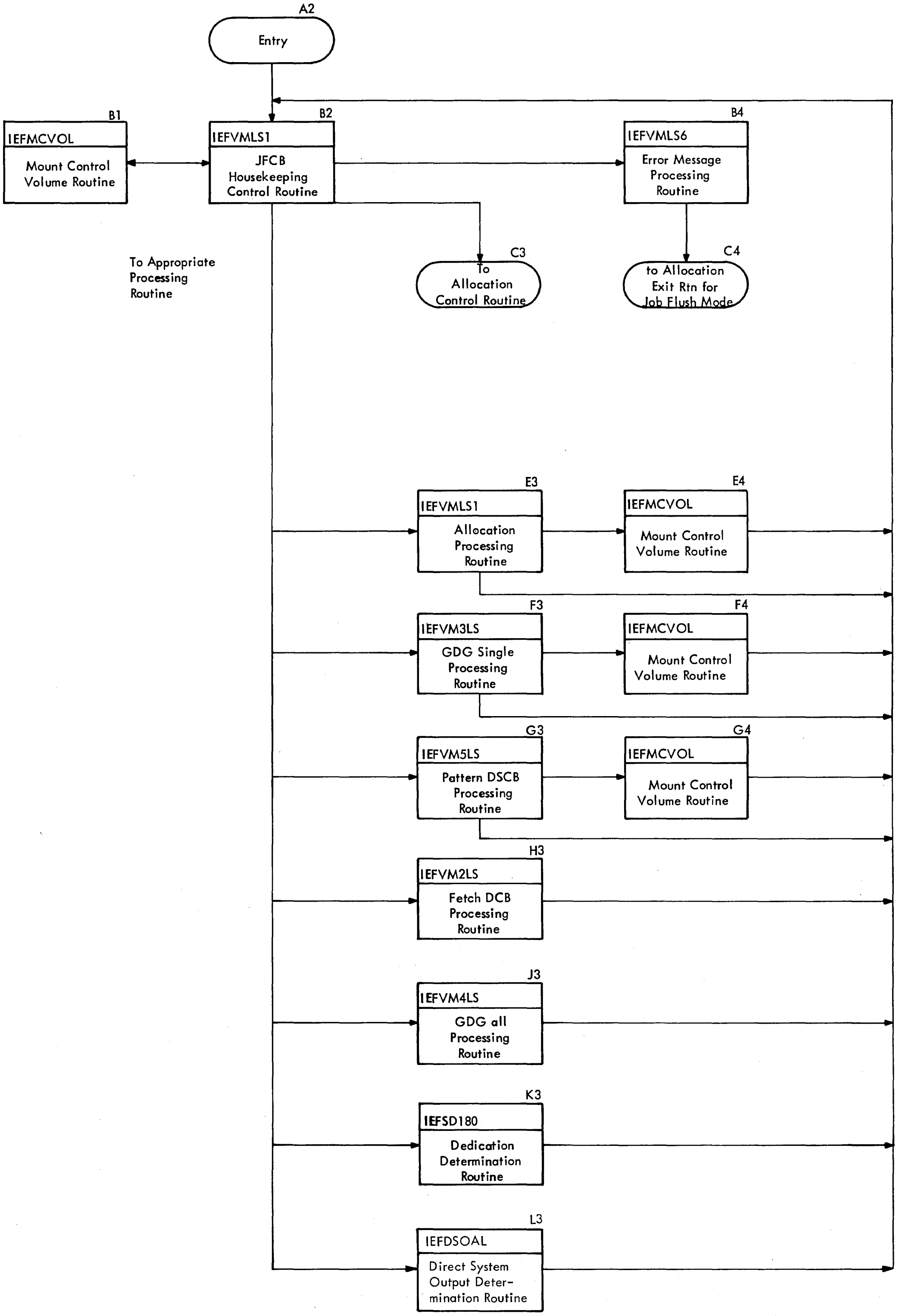

A2
Entry
B1
IEFMCVOL
Mount Control Volume Routine
B2
IEFVMLS1
JFCB Housekeeping Control Routine
B4
IEFVMLS6
Error Message Processing Routine
To Appropriate Processing Routine
C3
To Allocation Control Routine
C4
to Allocation Exit Rtn for Job Flush Mode
E3
IEFVMLS1
Allocation Processing Routine
E4
IEFMCVOL
Mount Control Volume Routine
F3
IEFVM3LS
GDG Single Processing Routine
F4
IEFMCVOL
Mount Control Volume Routine
G3
IEFVM5LS
Pattern DSCB Processing Routine
G4
IEFMCVOL
Mount Control Volume Routine
H3
IEFVM2LS
Fetch DCB Processing Routine
J3
IEFVM4LS
GDG all Processing Routine
K3
IEFSD180
Dedication Determination Routine
L3
IEFDSOAL
Direct System Output Determination Routine

Chart 10. JFCB Housekeeping Control Routine -- Part 1

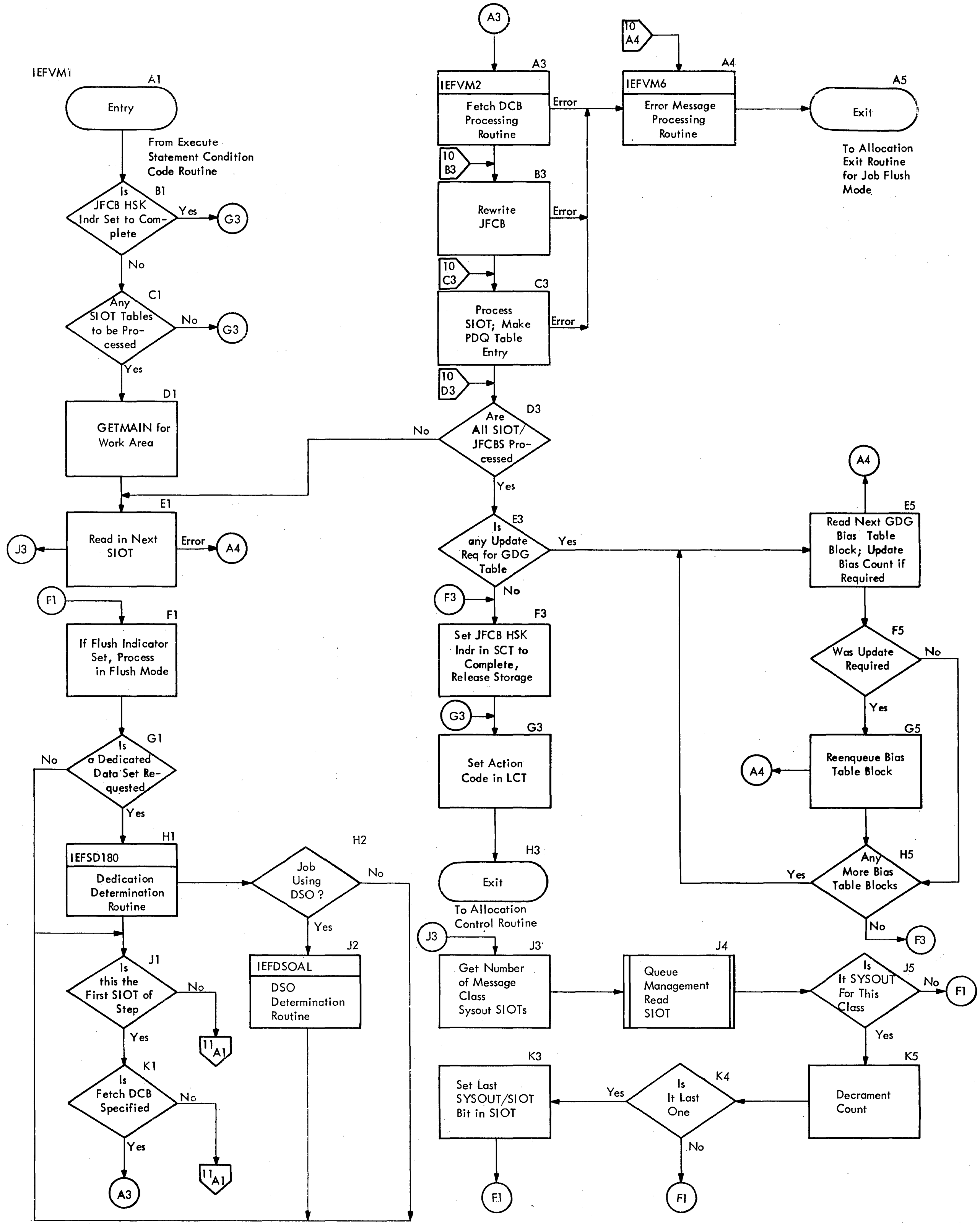

IEFVM1
A1
Entry
From Execute Statement Condition Code Routine
B1
Is JFCB HSK Indr Set to Complete
Yes
G3
No
C1
Any SIOT Tables to be Processed
No
G3
Yes
D1
GETMAIN for Work Area
E1
Read in Next SIOT
J3
Error
A4
F1
F1
If Flush Indicator Set, Process in Flush Mode
G1
Is a Dedicated Data Set Requested
No
Yes
H1
IEFSD180
Dedication Determination Routine
H2
Job Using DSO ?
No
Yes
J2
IEFDSOAL
DSO Determination Routine
J1
Is this the First SIOT of Step
No
11 A1
Yes
K1
Is Fetch DCB Specified
No
11 A1
Yes
A3
A3
A3
IEFVM2
Fetch DCB Processing Routine
Error
10 B3
B3
Rewrite JFCB
Error
10 C3
C3
Process SIOT; Make PDQ Table Entry
Error
10 D3
D3
Are All SIOT/JFCBS Processed
No
Yes
E3
Is any Update Req for GDG Table
Yes
No
F3
F3
Set JFCB HSK Indr in SCT to Complete, Release Storage
G3
G3
Set Action Code in LCT
H3
Exit
To Allocation Control Routine
J3
J3
Get Number of Message Class Sysout SIOTs
J4
Queue Management Read SIOT
K3
Set Last SYSOUT/SIOT Bit in SIOT
F1
K4
Is It Last One
Yes
No
F1
10 A4
A4
IEFVM6
Error Message Processing Routine
A5
Exit
To Allocation Exit Routine for Job Flush Mode
A4
E5
Read Next GDG Bias Table Block; Update Bias Count if Required
F5
Was Update Required
No
Yes
G5
Reenqueue Bias Table Block
A4
H5
Any More Bias Table Blocks
Yes
No
F3
J5
Is It SYSOUT For This Class
No
F1
Yes
K5
Decrement Count

Chart 11. JFCB Housekeeping Control Routine -- Part 2

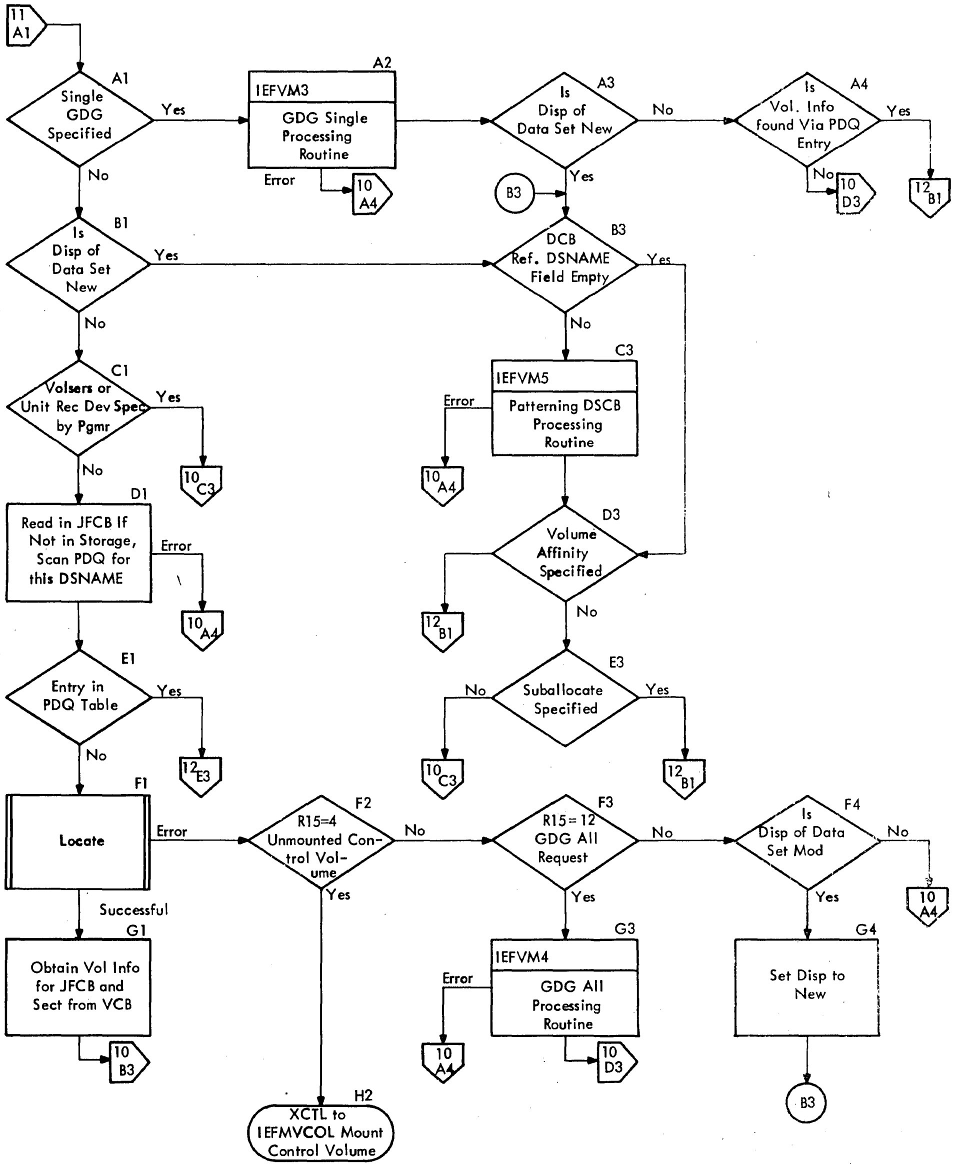
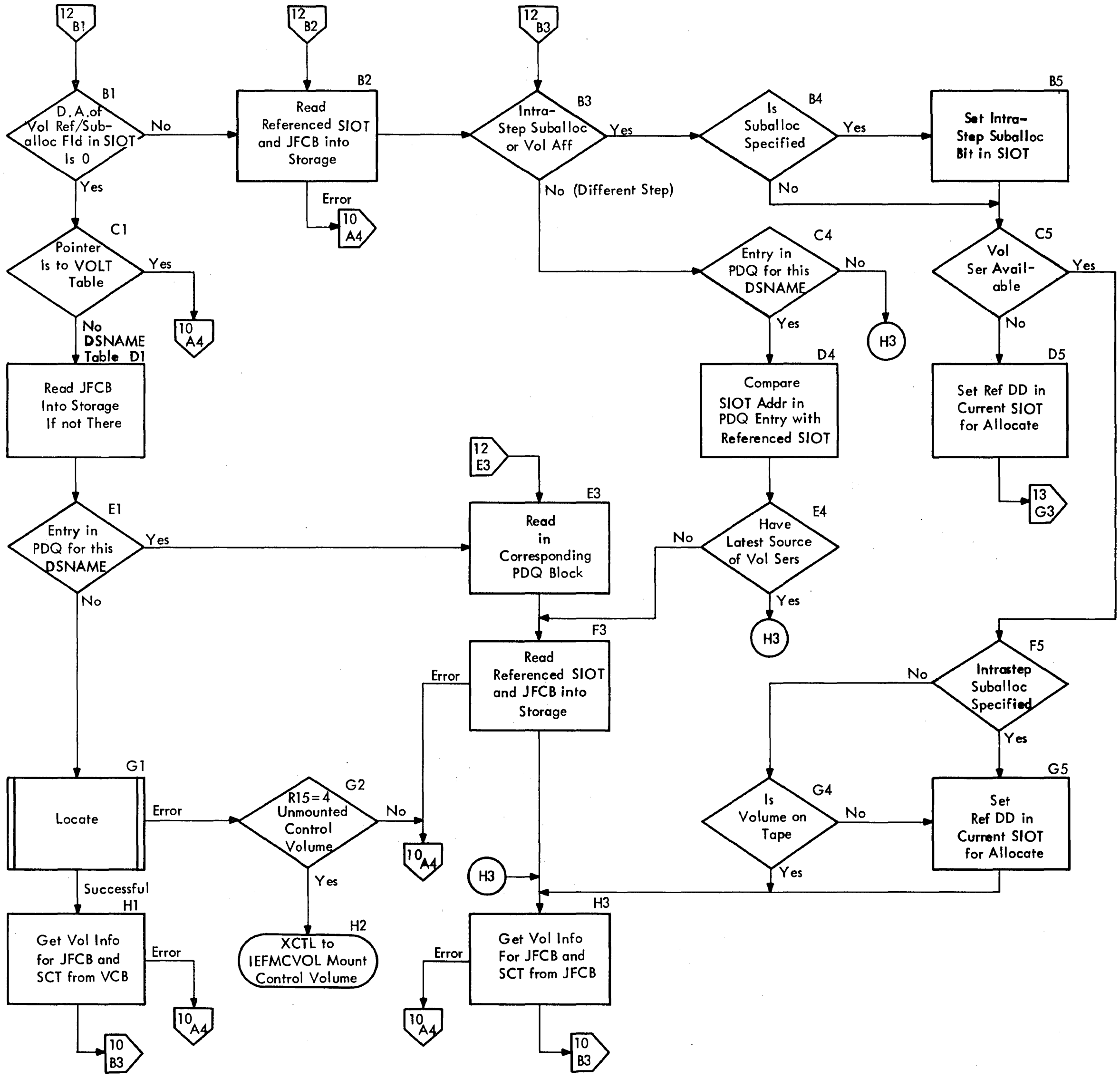


Chart 12. ALLOC Processing Routine



12 B1
B1
D.A. of Vol Ref/Sub-alloc Fld in SIOT Is 0
No
Yes
12 B2
B2
Read Referenced SIOT and JFCB into Storage
Error
10 A4
12 B3
B3
Intra-Step Suballoc or Vol Aff
Yes
No (Different Step)
B4
Is Suballoc Specified
Yes
No
B5
Set Intra-Step Suballoc Bit in SIOT
C1
Pointer Is to VOLT Table
Yes
10 A4
No DSNAME Table D1
Read JFCB Into Storage If not There
C4
Entry in PDQ for this DSNAME
No
H3
Yes
D4
Compare SIOT Addr in PDQ Entry with Referenced SIOT
C5
Vol Ser Available
Yes
No
D5
Set Ref DD in Current SIOT for Allocate
13 G3
12 E3
E3
Read in Corresponding PDQ Block
E1
Entry in PDQ for this DSNAME
Yes
No
E4
Have Latest Source of Vol Sers
No
Yes
H3
F3
Read Referenced SIOT and JFCB into Storage
Error
F5
Intrastep Suballoc Specified
No
Yes
G1
Locate
Error
G2
R15=4 Unmounted Control Volume
No
10 A4
Yes
G4
Is Volume on Tape
No
Yes
G5
Set Ref DD in Current SIOT for Allocate
H3
Successful
H1
Get Vol Info for JFCB and SCT from VCB
Error
10 A4
10 B3
H2
XCTL to IEFMCVOL Mount Control Volume
H3
Get Vol Info For JFCB and SCT from JFCB
Error
10 A4
10 B3

All Entries/Exits are from/to the JFCB Housekeeping Control Routine

Chart 13. Dedication Determination Routine

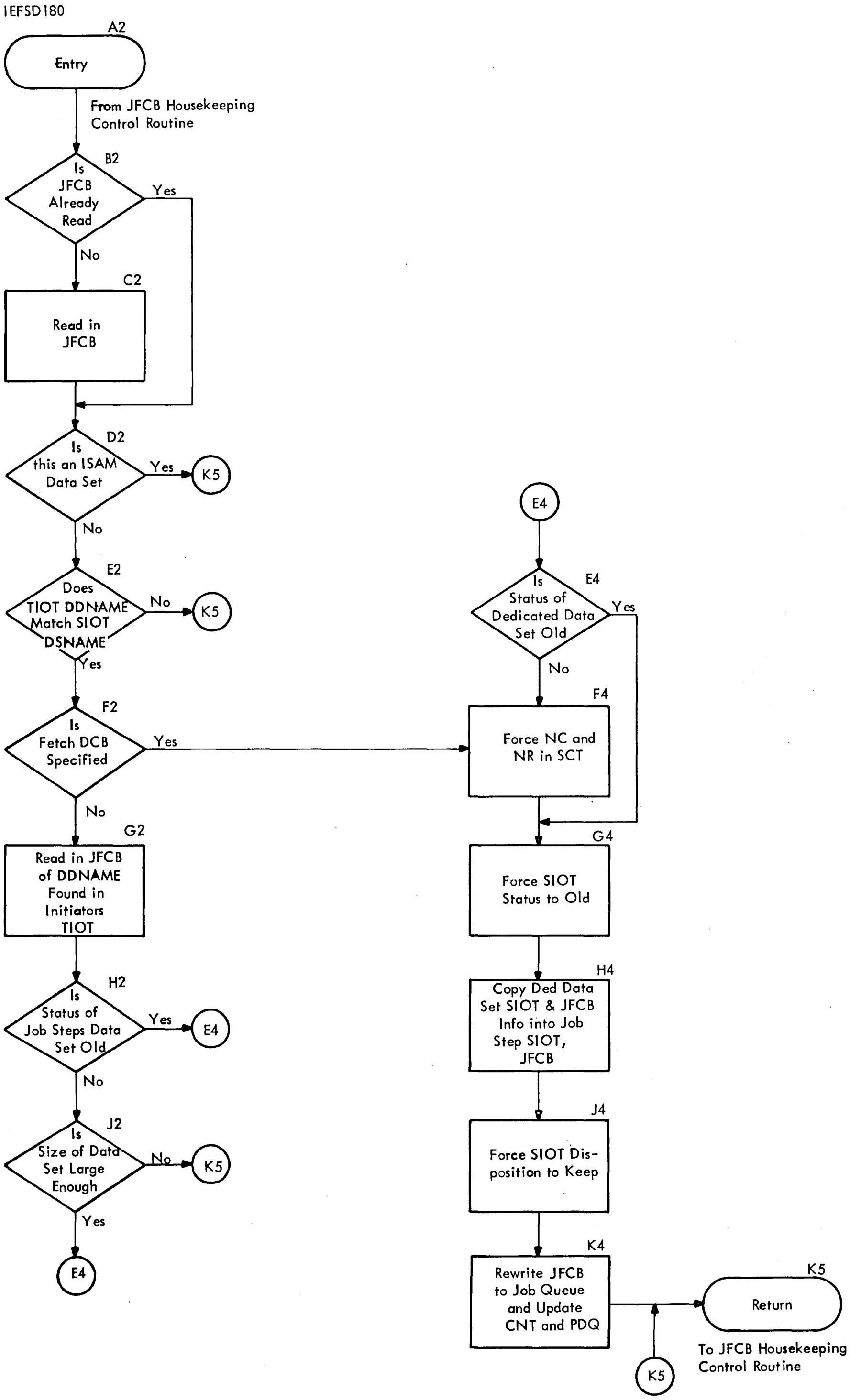

IEFSD180
A2
Entry
From JFCB Housekeeping Control Routine
B2
Is JFCB Already Read
Yes
No
C2
Read in JFCB
D2
Is this an ISAM Data Set
Yes
K5
No
E2
Does TIOT DDNAME Match SIOT DSNAME
No
K5
Yes
F2
Is Fetch DCB Specified
Yes
No
G2
Read in JFCB of DDNAME Found in Initiators TIOT
H2
Is Status of Job Steps Data Set Old
Yes
E4
No
J2
Is Size of Data Set Large Enough
No
K5
Yes
E4
E4
E4
Is Status of Dedicated Data Set Old
Yes
No
F4
Force NC and NR in SCT
G4
Force SIOT Status to Old
H4
Copy Ded Data Set SIOT & JFCB Info into Job Step SIOT, JFCB
J4
Force SIOT Dis-position to Keep
K4
Rewrite JFCB to Job Queue and Update CNT and PDQ
K5
Return
To JFCB Housekeeping Control Routine
K5

Chart 14. Fetch DCB Processing Routine

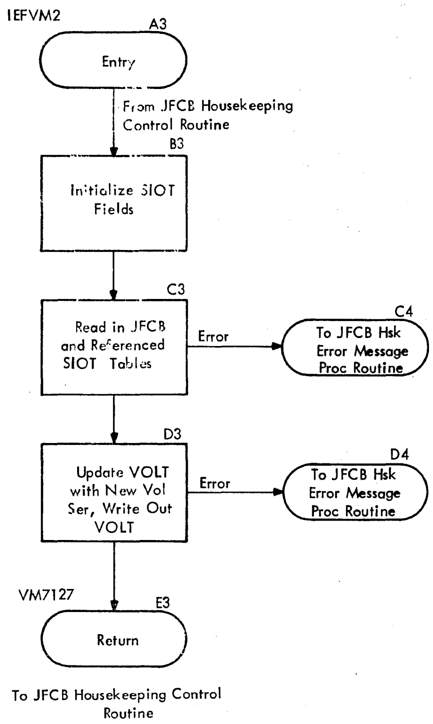

IEFVM2
A3
Entry
From JFCB Housekeeping Control Routine
B3
Initialize SIOT Fields
C3
Read in JFCB and Referenced SIOT Tables
Error
C4
To JFCB Hsk Error Message Proc Routine
D3
Update VOLT with New Vol Ser, Write Out VOLT
Error
D4
To JFCB Hsk Error Message Proc Routine
VM7127
E3
Return
To JFCB Housekeeping Control Routine

Chart 15. GDG Single Processing Routine

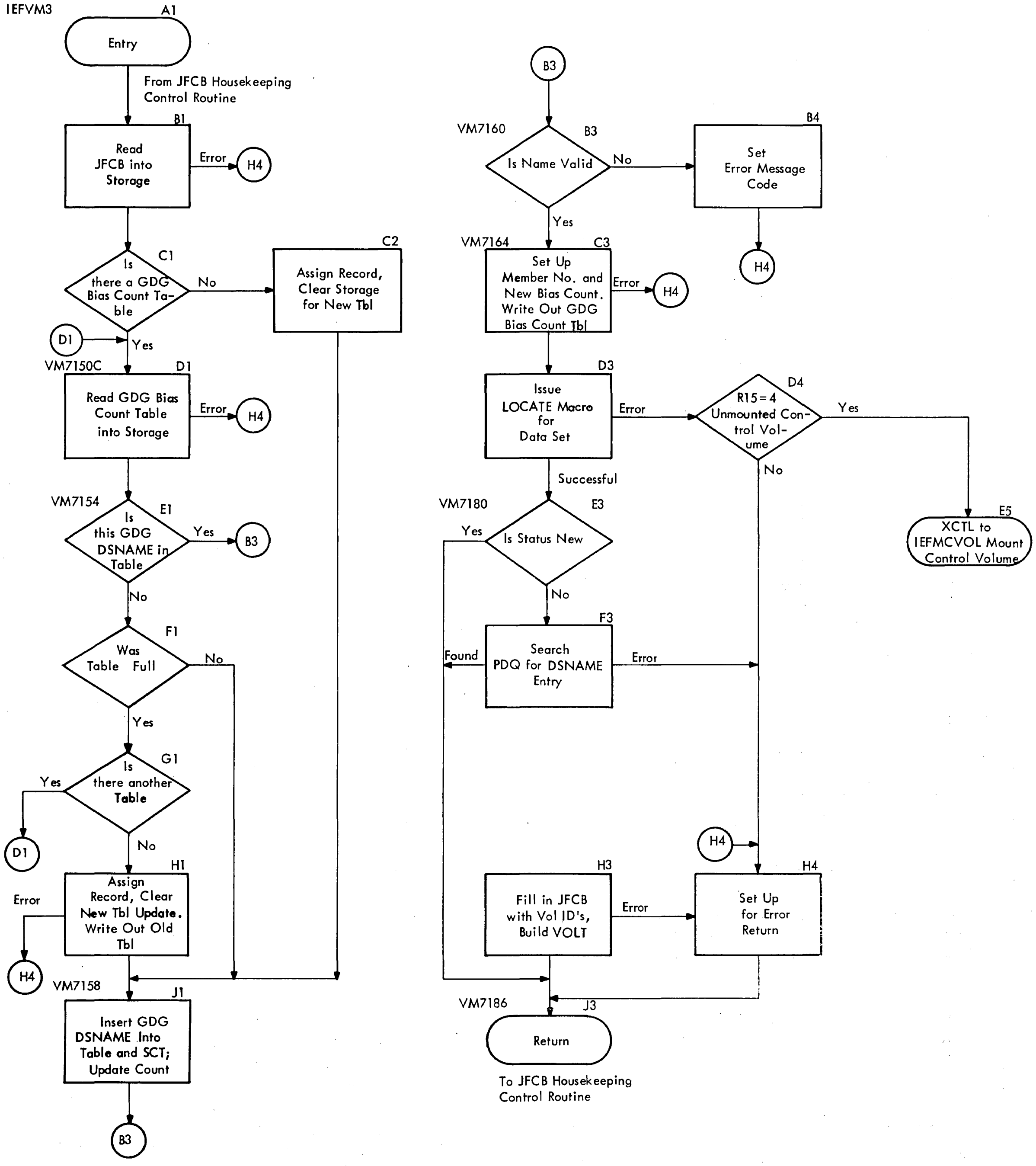
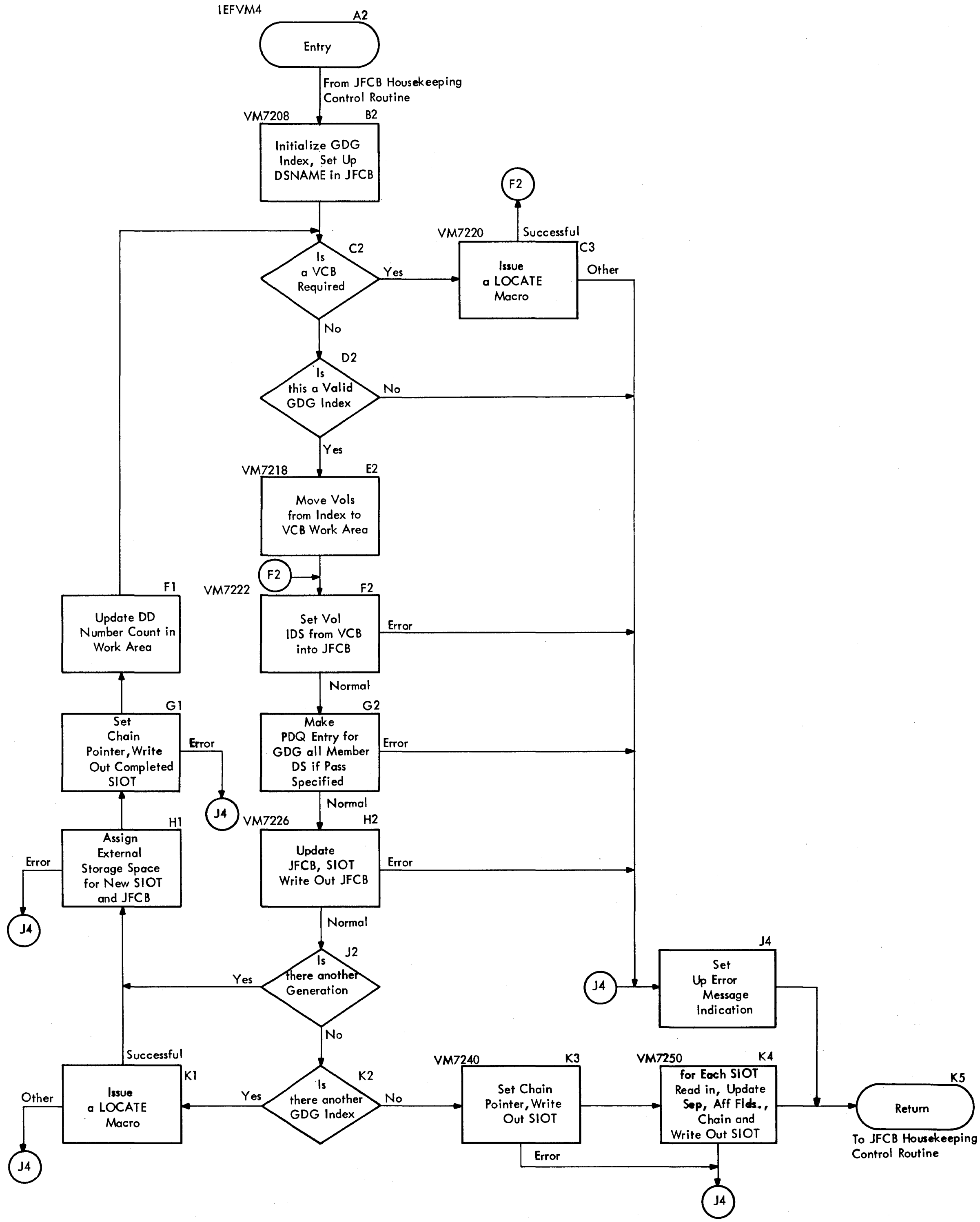

IEFVM3
A1
Entry
From JFCB Housekeeping Control Routine
B1
Read JFCB into Storage
Error
H4
C1
Is there a GDG Bias Count Table
No
C2
Assign Record, Clear Storage for New Tbl
D1
Yes
VM7150C
D1
Read GDG Bias Count Table into Storage
Error
H4
VM7154
E1
Is this GDG DSNAME in Table
Yes
B3
No
F1
Was Table Full
No
Yes
G1
Is there another Table
Yes
D1
No
H1
Assign Record, Clear New Tbl Update. Write Out Old Tbl
Error
H4
VM7158
J1
Insert GDG DSNAME Into Table and SCT; Update Count
B3
B3
VM7160
B3
Is Name Valid
No
B4
Set Error Message Code
H4
Yes
VM7164
C3
Set Up Member No. and New Bias Count. Write Out GDG Bias Count Tbl
Error
H4
D3
Issue LOCATE Macro for Data Set
Error
D4
R15=4 Unmounted Control Volume
Yes
No
E5
XCTL to IEFMCVOL Mount Control Volume
Successful
VM7180
E3
Yes
Is Status New
No
F3
Found
Search PDQ for DSNAME Entry
Error
H4
H3
Fill in JFCB with Vol ID's, Build VOLT
Error
H4
Set Up for Error Return
VM7186
J3
Return
To JFCB Housekeeping Control Routine

Chart 16. GDG All Processing Routine



IEFVM4
A2
Entry
From JFCB Housekeeping Control Routine
VM7208
B2
Initialize GDG Index, Set Up DSNAME in JFCB
F2
VM7220
Successful
C2
Is a VCB Required
Yes
Issue a LOCATE Macro
C3
Other
No
D2
Is this a Valid GDG Index
No
Yes
VM7218
E2
Move Vols from Index to VCB Work Area
F2
F1
Update DD Number Count in Work Area
VM7222
F2
Set Vol IDS from VCB into JFCB
Error
Normal
G1
Set Chain Pointer, Write Out Completed SIOT
Error
J4
G2
Make PDQ Entry for GDG all Member DS if Pass Specified
Error
Normal
VM7226
H2
H1
Assign External Storage Space for New SIOT and JFCB
Error
J4
Update JFCB, SIOT Write Out JFCB
Error
Normal
J2
Is there another Generation
Yes
J4
J4
Set Up Error Message Indication
No
Successful
K1
Issue a LOCATE Macro
Other
J4
K2
Is there another GDG Index
Yes
No
VM7240
K3
Set Chain Pointer, Write Out SIOT
Error
VM7250
K4
for Each SIOT Read in, Update Sep, Aff Flds., Chain and Write Out SIOT
J4
K5
Return
To JFCB Housekeeping Control Routine

Chart 17. Patterning DSCB Processing Routine

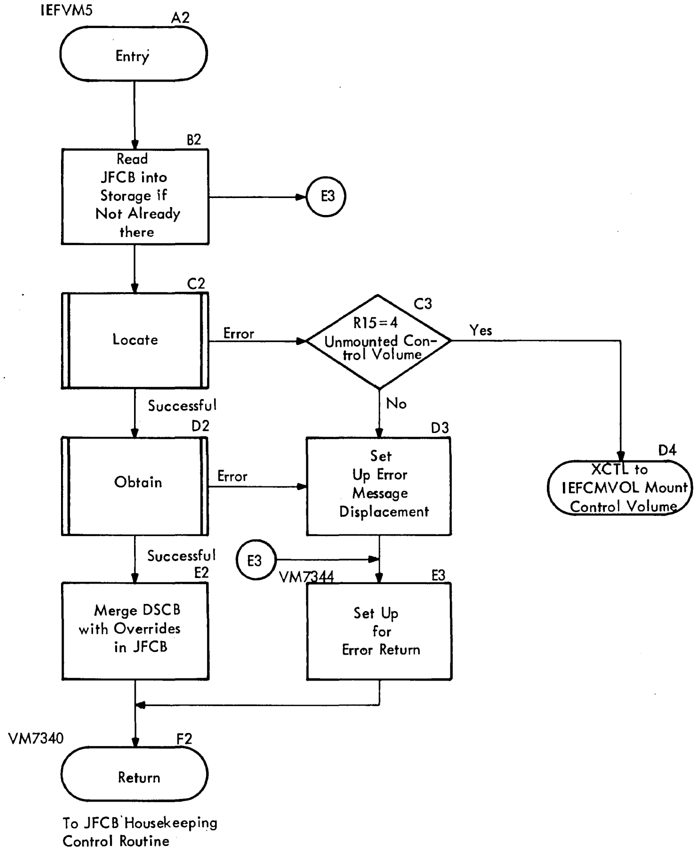

IEFVM5
A2
Entry
B2
Read JFCB into Storage if Not Already there
E3
C2
Locate
Error
C3
R15=4 Unmounted Control Volume
Yes
No
Successful
D2
Obtain
Error
D3
Set Up Error Message Displacement
D4
XCTL to IEFCMVOL Mount Control Volume
Successful
E2
E3
VM7344
E3
Merge DSCB with Overrides in JFCB
Set Up for Error Return
VM7340
F2
Return
To JFCB Housekeeping Control Routine

Chart 18. Mount Control Volume Routine

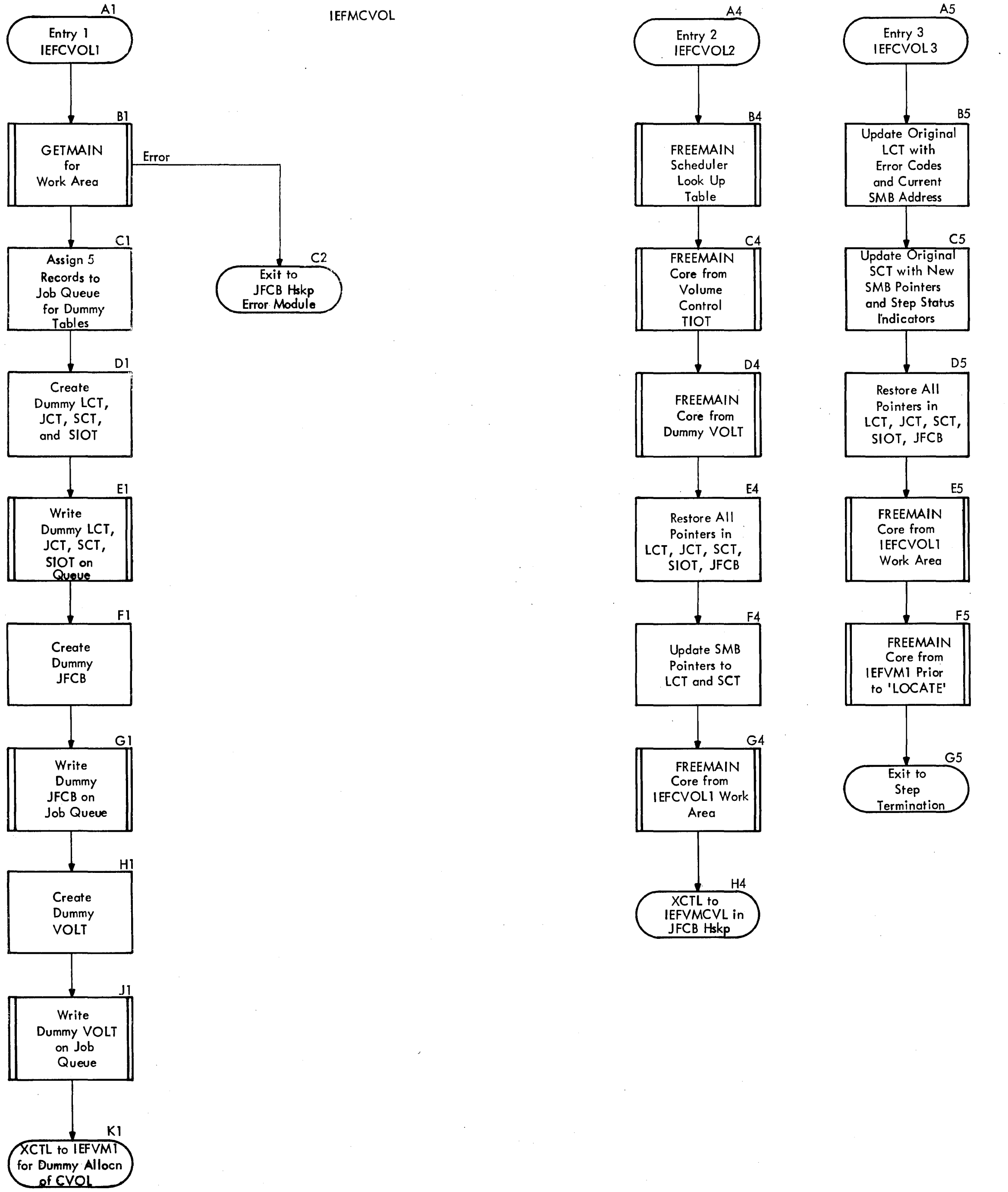
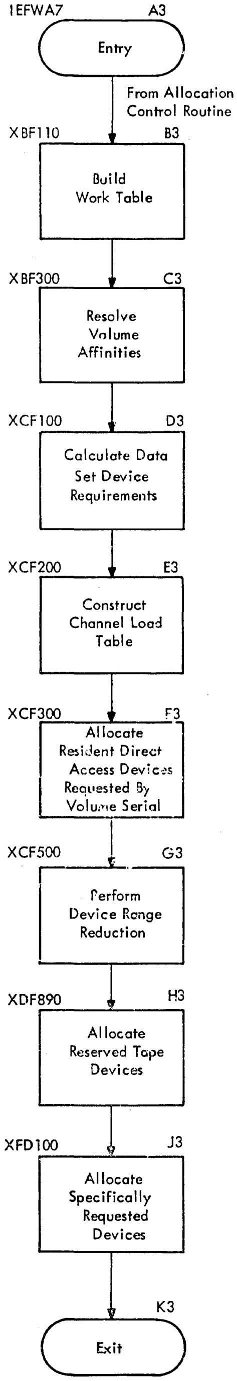

IEFMCVOL
A1
Entry 1
IEFCVOL1
B1
GETMAIN for Work Area
Error
C1
Assign 5 Records to Job Queue for Dummy Tables
C2
Exit to JFCB Hskp Error Module
D1
Create Dummy LCT, JCT, SCT, and SIOT
E1
Write Dummy LCT, JCT, SCT, SIOT on Queue
F1
Create Dummy JFCB
G1
Write Dummy JFCB on Job Queue
H1
Create Dummy VOLT
J1
Write Dummy VOLT on Job Queue
K1
XCTL to IEFVM1 for Dummy Allocn of CVOL
A4
Entry 2
IEFCVOL2
B4
FREEMAIN Scheduler Look Up Table
C4
FREEMAIN Core from Volume Control TIOT
D4
FREEMAIN Core from Dummy VOLT
E4
Restore All Pointers in LCT, JCT, SCT, SIOT, JFCB
F4
Update SMB Pointers to LCT and SCT
G4
FREEMAIN Core from IEFCVOL1 Work Area
H4
XCTL to IEFVMCVL in JFCB Hskp
A5
Entry 3
IEFCVOL3
B5
Update Original LCT with Error Codes and Current SMB Address
C5
Update Original SCT with New SMB Pointers and Step Status Indicators
D5
Restore All Pointers in LCT, JCT, SCT, SIOT, JFCB
E5
FREEMAIN Core from IEFCVOL1 Work Area
F5
FREEMAIN Core from IEFVM1 Prior to 'LOCATE'
G5
Exit to Step Termination

Chart 19. Demand Allocation Routine



IEFWA7
A3
Entry
From Allocation Control Routine
XBF110
B3
Build Work Table
XBF300
C3
Resolve Volume Affinities
XCF100
D3
Calculate Data Set Device Requirements
XCF200
E3
Construct Channel Load Table
XCF300
F3
Allocate Resident Direct Access Devices Requested By Volume Serial
XCF500
G3
Perform Device Range Reduction
XDF890
H3
Allocate Reserved Tape Devices
XFD100
J3
Allocate Specifically Requested Devices
K3
Exit

Exits are to the Decision Allocation Routine (Chart 20),If Allocation is not Complete.

To the TIOT Construction Routine (Chart 21),If Allocation is Complete.

Chart 20. Decision Allocation Routine

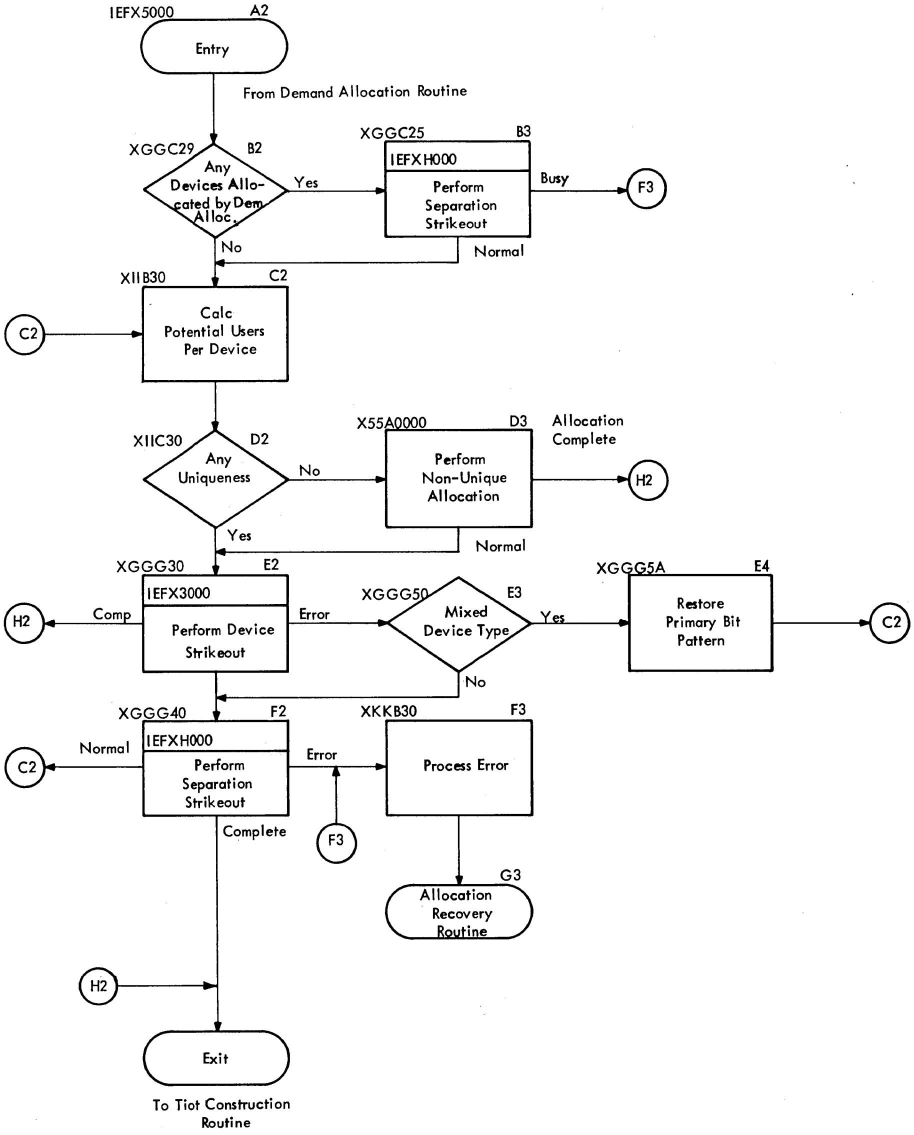
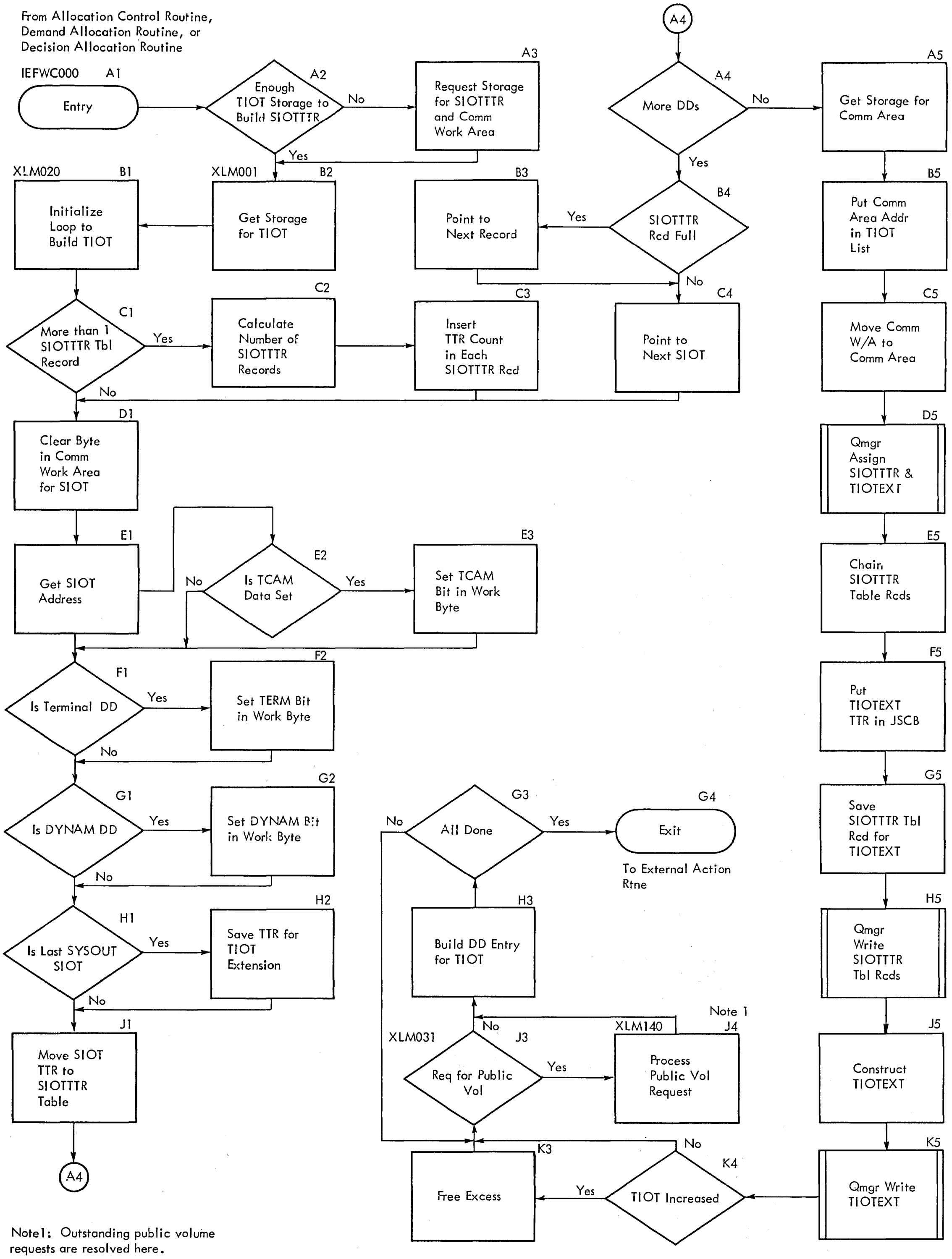
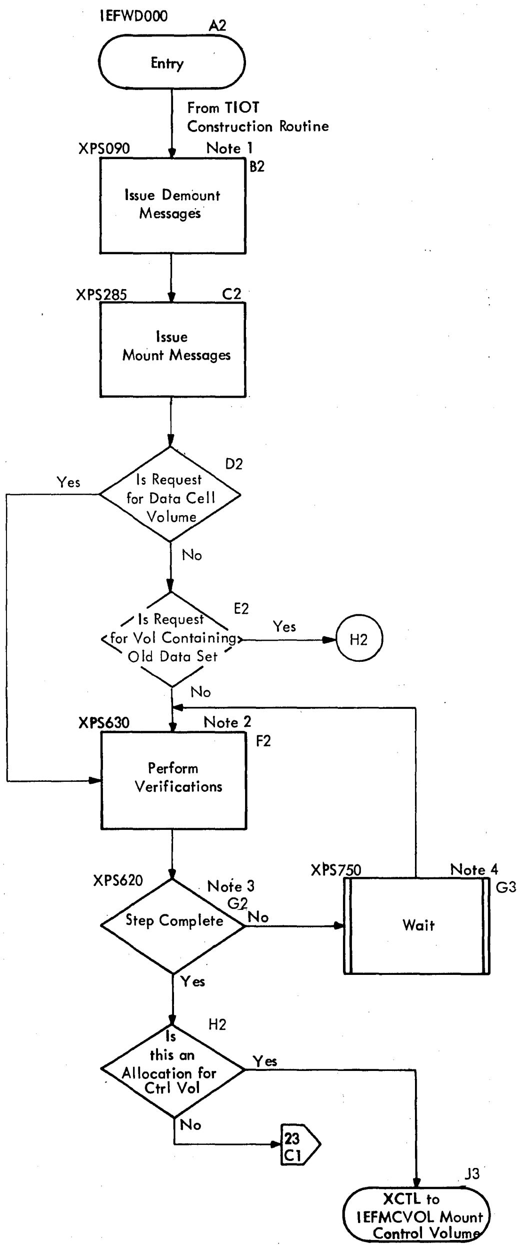


Chart 21. TIOT Construction Routine


From Allocation Control Routine,
Demand Allocation Routine, or
Decision Allocation Routine
IEFWC000 A1
Entry
A2
Enough TIOT Storage to Build SIOTTTR
No
Yes
A3
Request Storage for SIOTTTR and Comm Work Area
A4
A4
More DDs
No
Yes
A5
Get Storage for Comm Area
XLM020 B1
Initialize Loop to Build TIOT
XLM001 B2
Get Storage for TIOT
B3
Point to Next Record
B4
SIOTTTR Rcd Full
Yes
No
B5
Put Comm Area Addr in TIOT List
C1
More than 1 SIOTTTR Tbl Record
Yes
No
C2
Calculate Number of SIOTTTR Records
C3
Insert TTR Count in Each SIOTTTR Rcd
C4
Point to Next SIOT
C5
Move Comm W/A to Comm Area
D1
Clear Byte in Comm Work Area for SIOT
D5
Qmgr Assign SIOTTTR & TIOTEXT
E1
Get SIOT Address
E2
Is TCAM Data Set
No
Yes
E3
Set TCAM Bit in Work Byte
E5
Chain SIOTTTR Table Rcds
F1
Is Terminal DD
Yes
No
F2
Set TERM Bit in Work Byte
F5
Put TIOTEXT TTR in JSCB
G1
Is DYNAM DD
Yes
No
G2
Set DYNAM Bit in Work Byte
G3
All Done
No
Yes
G4
Exit
To External Action Rtne
G5
Save SIOTTTR Tbl Rcd for TIOTEXT
H1
Is Last SYSOUT SIOT
Yes
No
H2
Save TTR for TIOT Extension
H3
Build DD Entry for TIOT
H5
Qmgr Write SIOTTTR Tbl Rcds
J1
Move SIOT TTR to SIOTTTR Table
XLM031 J3
Req for Public Vol
No
Yes
Note 1
XLM140 J4
Process Public Vol Request
J5
Construct TIOTEXT
A4
K3
Free Excess
K4
TIOT Increased
Yes
No
K5
Qmgr Write TIOTEXT

Note 1: Outstanding public volume requests are resolved here.

Chart 22. External Action Routine



Note 1: Includes a Scan to Determine If the Required Volumes are Mounted on Unallocated Devices.

Note 2: Unload Commands from Master Scheduler are Honored. UCBs are Updated If Required.

Note 3: The Step is Complete When all Setup Messages Have Been Issued and Verification Has Been Performed. Counts in the SCT Control this Mechanism.

Note 4: Either of Two Events is Waited Upon. Issuance of a Cancel Command or a Device Being Made Ready.

Chart 23. Space Request and TIOT Compression Routines

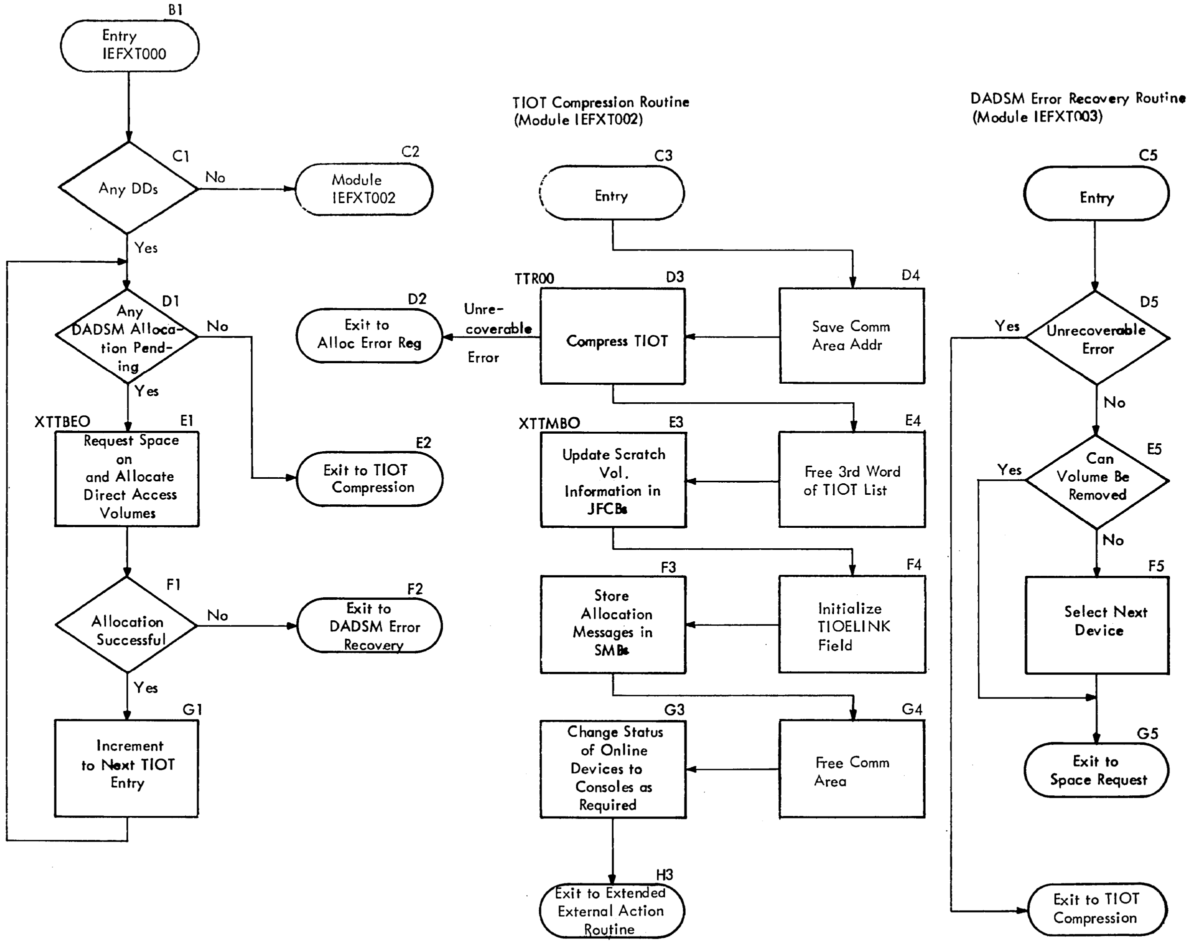

B1
Entry IEFXT000
C1
Any DDs
No
Yes
C2
Module IEFXT002
TIOT Compression Routine (Module IEFXT002)
C3
Entry
DADSM Error Recovery Routine (Module IEFXT003)
C5
Entry
D1
Any DADSM Allocation Pending
No
Yes
D2
Exit to Alloc Error Reg
Unrecoverable Error
TTR00
D3
Compress TIOT
D4
Save Comm Area Addr
D5
Unrecoverable Error
Yes
No
XTTBE0
E1
Request Space on and Allocate Direct Access Volumes
E2
Exit to TIOT Compression
XTTMB0
E3
Update Scratch Vol. Information in JFCBs
E4
Free 3rd Word of TIOT List
E5
Can Volume Be Removed
Yes
No
F1
Allocation Successful
No
Yes
F2
Exit to DADSM Error Recovery
F3
Store Allocation Messages in SMBs
F4
Initialize TIOELINK Field
F5
Select Next Device
G1
Increment to Next TIOT Entry
G3
Change Status of Online Devices to Consoles as Required
G4
Free Comm Area
G5
Exit to Space Request
H3
Exit to Extended External Action Routine
Exit to TIOT Compression

Chart 24. Extended External Action Routine

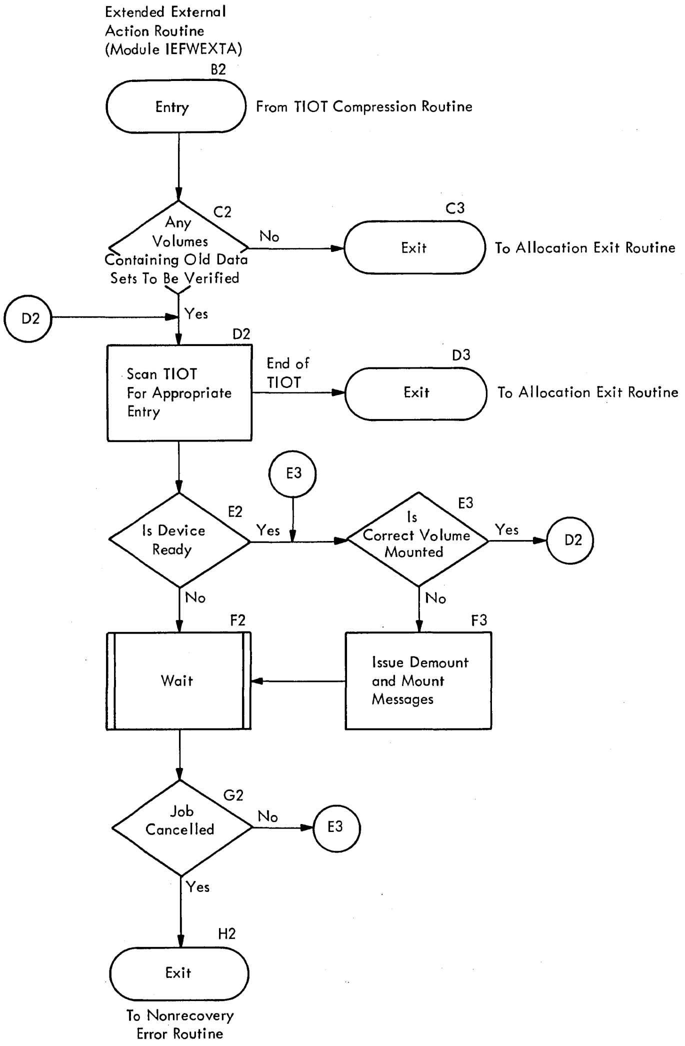
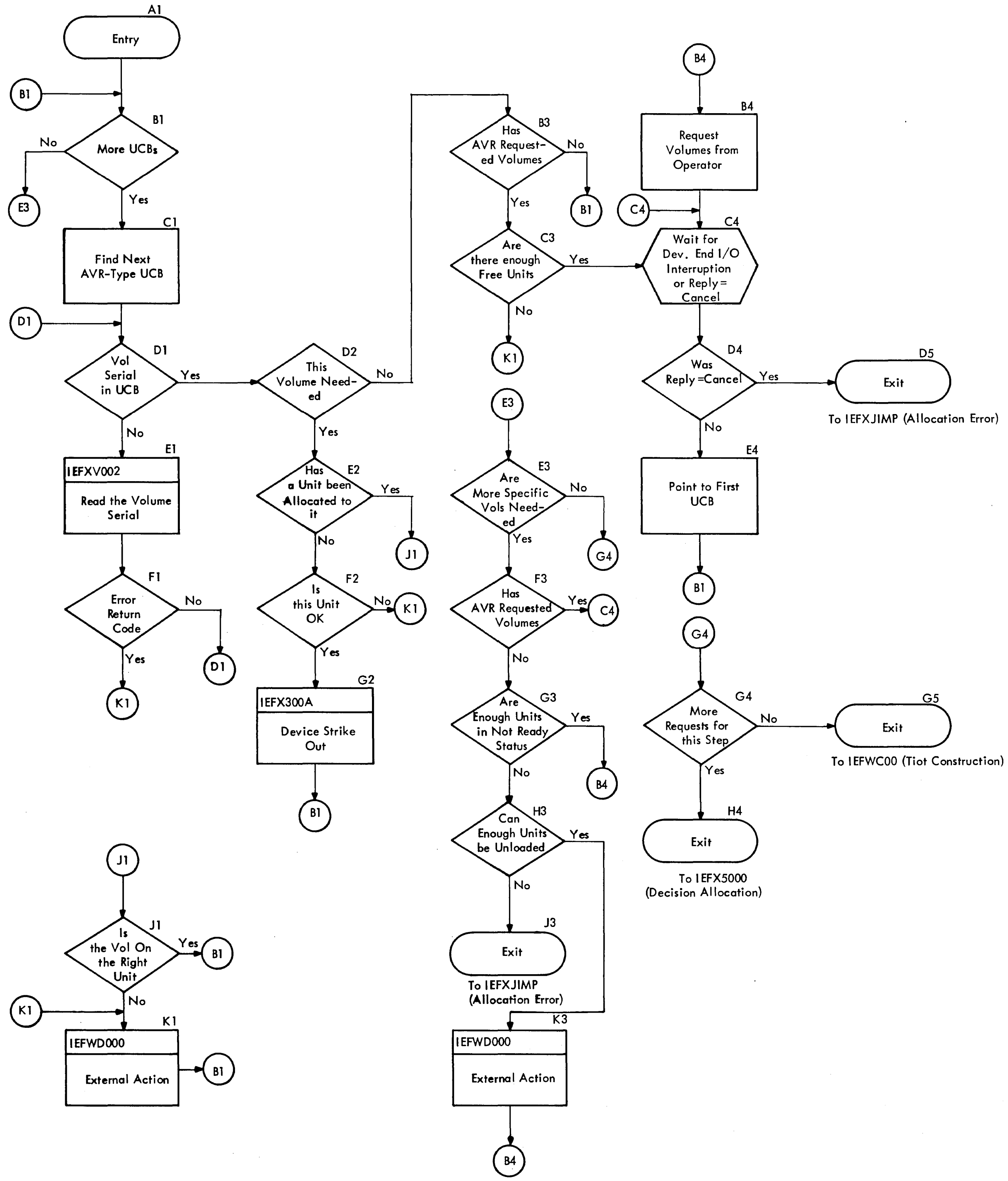

Extended External
Action Routine
(Module IEFWEXTA)
B2
Entry
From TIOT Compression Routine
C2
Any Volumes Containing Old Data Sets To Be Verified
No
C3
Exit
To Allocation Exit Routine
D2
Yes
D2
Scan TIOT For Appropriate Entry
End of TIOT
D3
Exit
To Allocation Exit Routine
E3
E2
Is Device Ready
Yes
E3
Is Correct Volume Mounted
Yes
D2
No
No
F2
F3
Wait
Issue Demount and Mount Messages
G2
Job Cancelled
No
E3
Yes
H2
Exit
To Nonrecovery Error Routine

Chart 25. Automatic Volume Recognition (AVR) Routine



A1
Entry
B1
B1
More UCBs
No
E3
Yes
C1
Find Next AVR-Type UCB
D1
D1
Vol Serial in UCB
Yes
No
E1
IEFXV002
Read the Volume Serial
F1
Error Return Code
No
D1
Yes
K1
D2
This Volume Needed
No
Yes
E2
Has a Unit been Allocated to it
Yes
J1
No
F2
Is this Unit OK
No
K1
Yes
G2
IEFX300A
Device Strike Out
B1
J1
J1
Is the Vol On the Right Unit
Yes
B1
No
K1
K1
IEFWD000
External Action
B1
B3
Has AVR Requested Volumes
No
B1
Yes
C3
Are there enough Free Units
Yes
No
K1
E3
E3
Are More Specific Vols Needed
No
G4
Yes
F3
Has AVR Requested Volumes
Yes
C4
No
G3
Are Enough Units in Not Ready Status
Yes
B4
No
H3
Can Enough Units be Unloaded
Yes
No
J3
Exit
To IEFXJIMP (Allocation Error)
K3
IEFWD000
External Action
B4
B4
B4
Request Volumes from Operator
C4
C4
Wait for Dev. End I/O Interruption or Reply = Cancel
D4
Was Reply =Cancel
Yes
D5
Exit
To IEFXJIMP (Allocation Error)
No
E4
Point to First UCB
B1
G4
G4
More Requests for this Step
No
G5
Exit
To IEFWC00 (Tiot Construction)
Yes
H4
Exit
To IEFX5000 (Decision Allocation)

Chart 26. Job Statement Condition Code Processing Routine

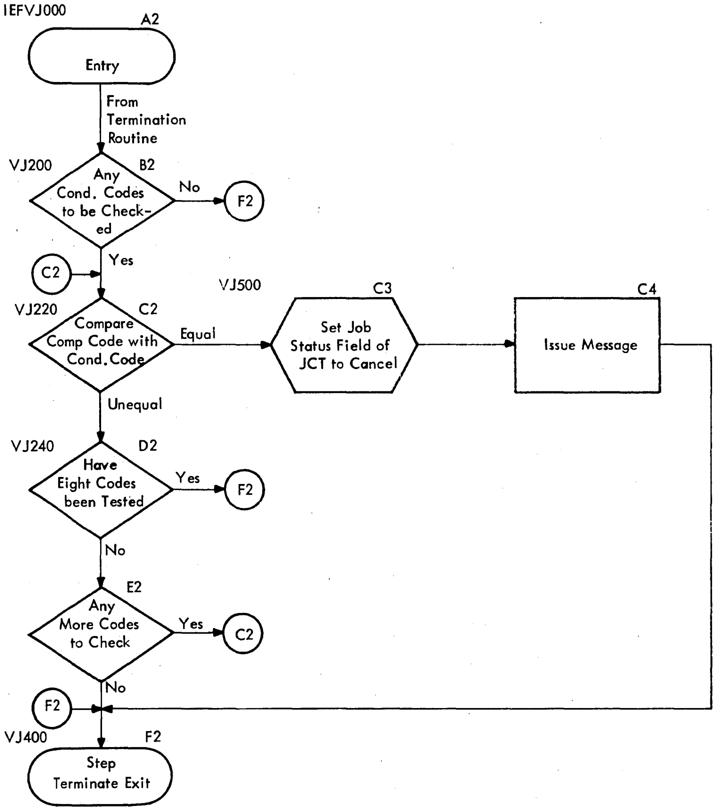
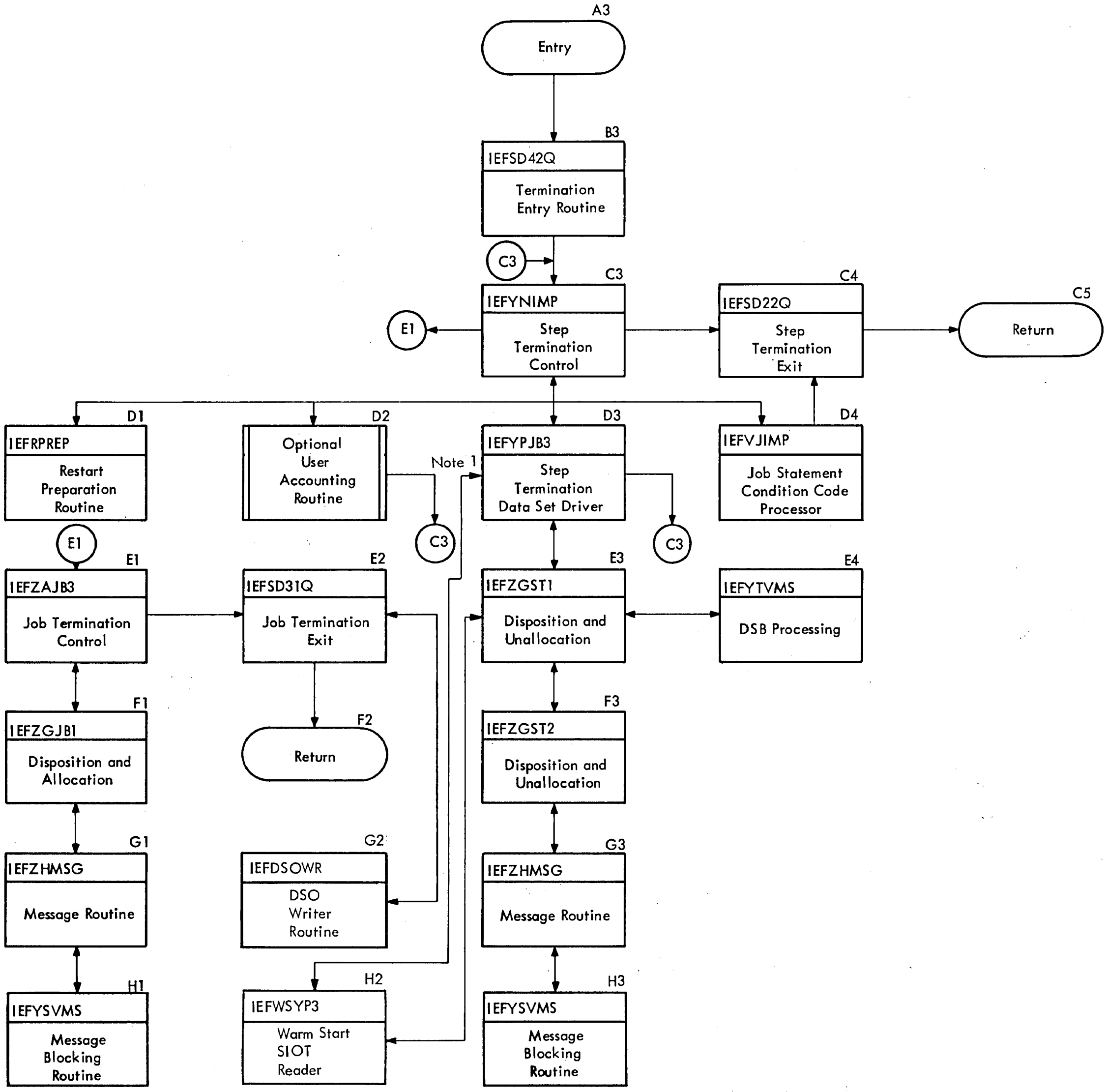
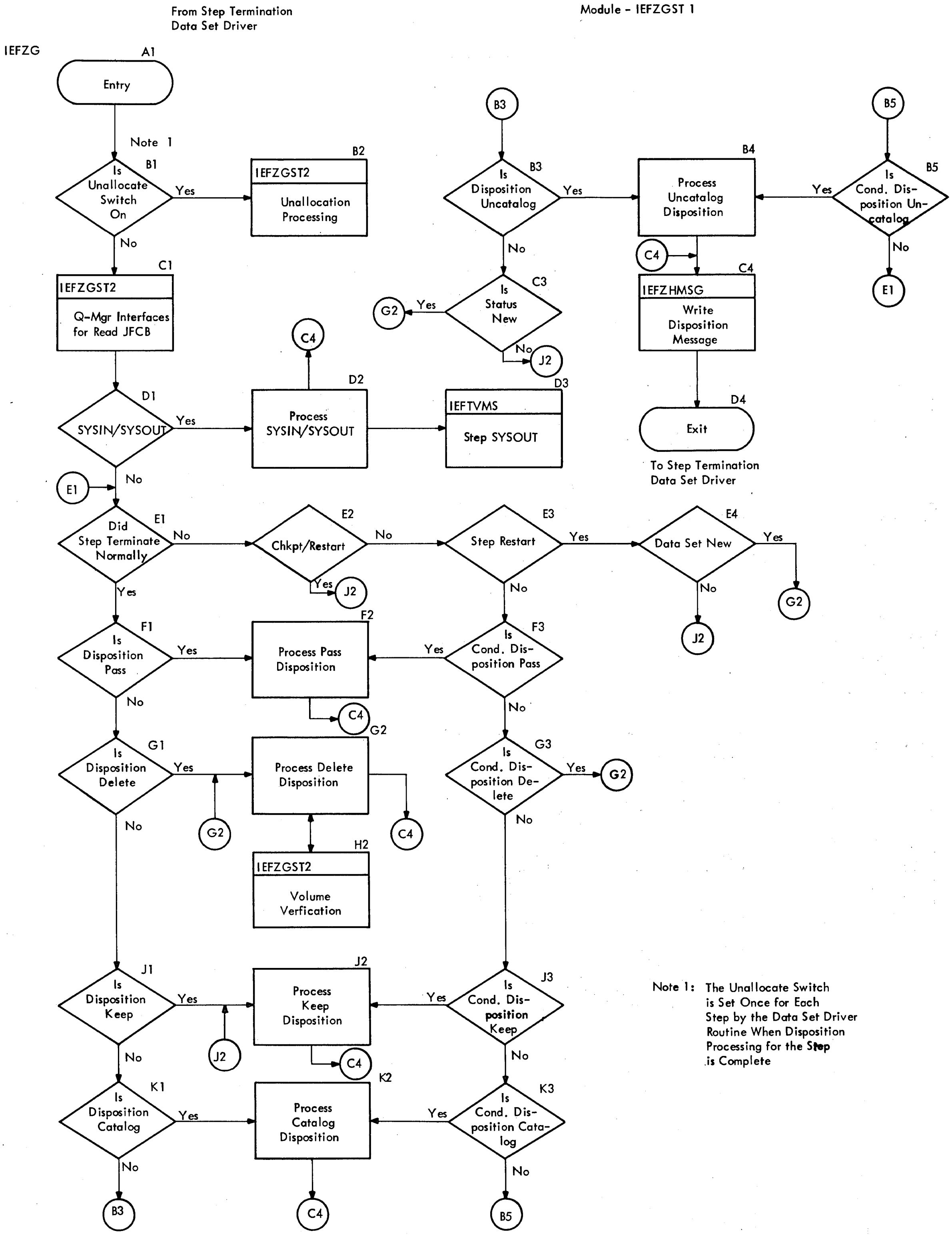


Chart 27. Termination Routine



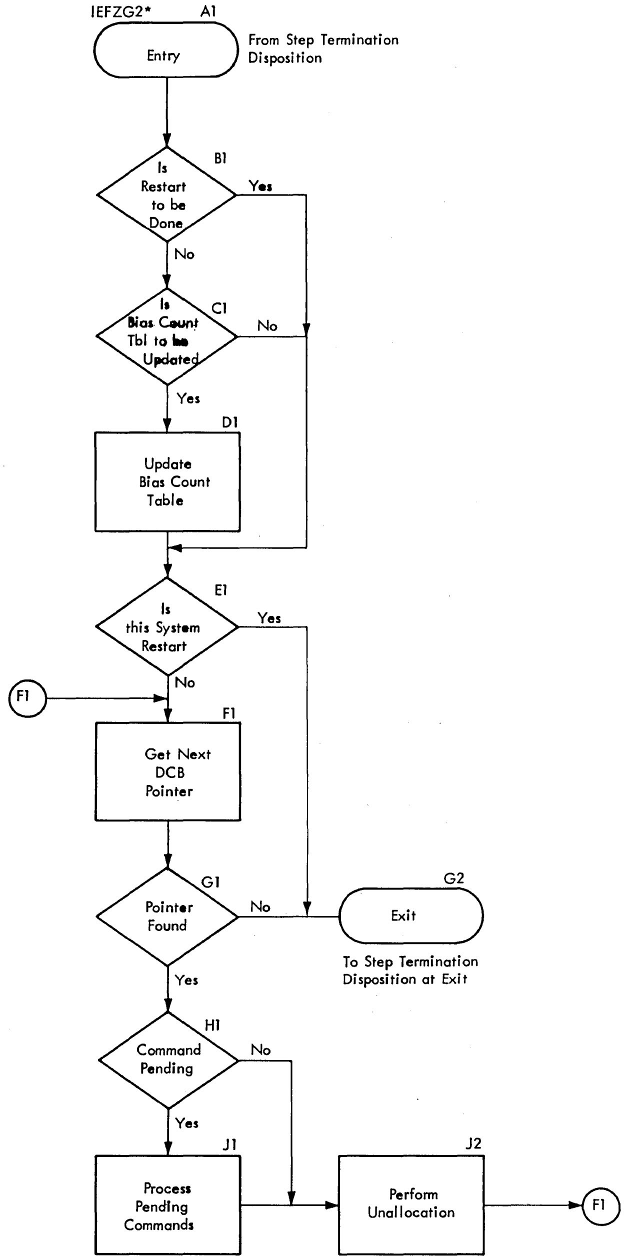
Note 1. If Warm Start, box D3 goes to box E3 via box H2, box E3 returns to H2. Box H2 makes the return to box D3.

Chart 28. Step Termination -- Disposition Routine



Note 1: The Unallocate Switch
is Set Once for Each
Step by the Data Set Driver
Routine When Disposition
Processing for the Step
is Complete

Chart 29. Step Termination -- Unallocation Routine



* There are Subroutines Present in this Module Which are Common to IEFZGST1 and IEFZGST2

Chart 30. Job Termination Routine

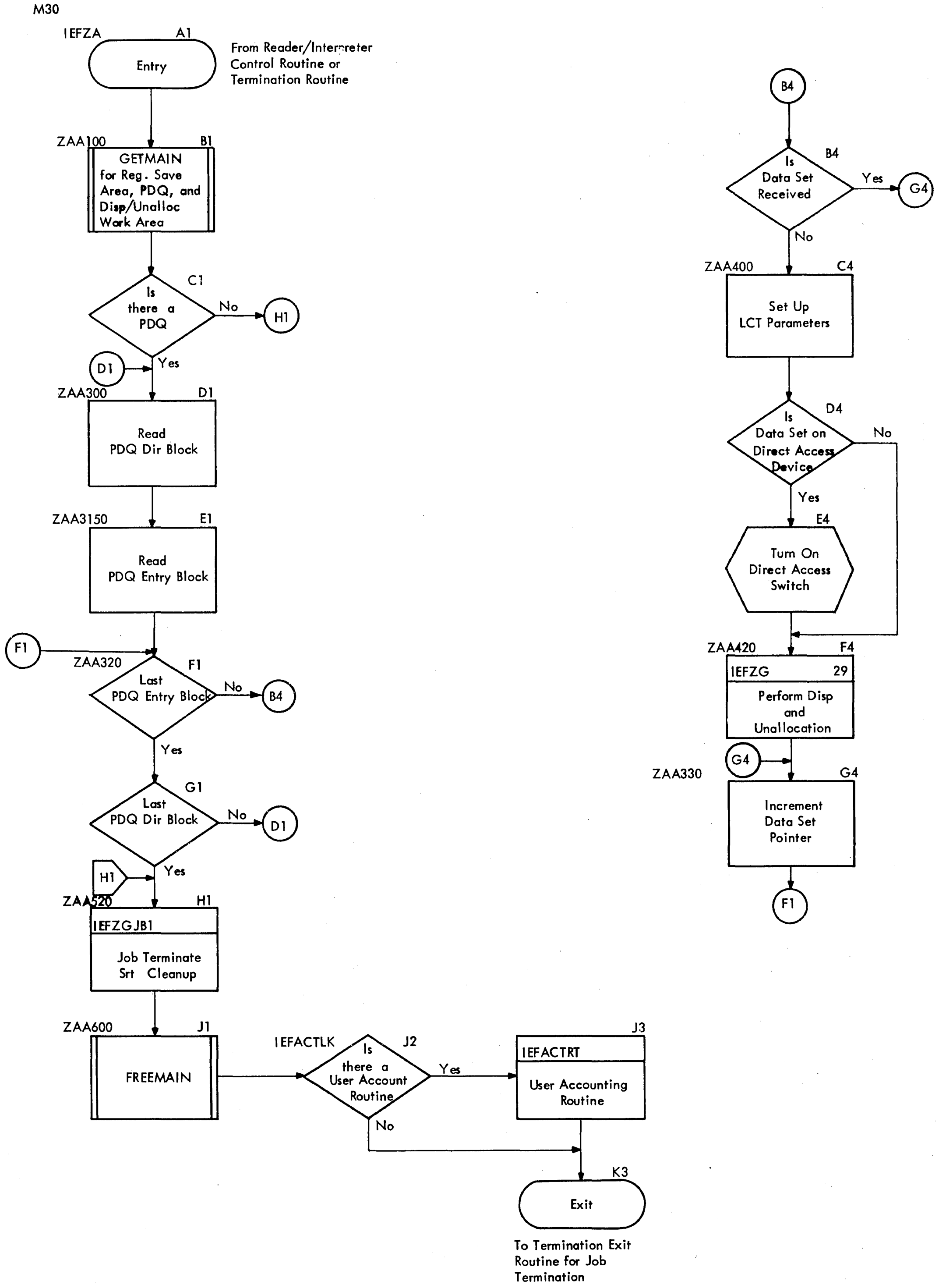


Chart 31. Job Termination -- Disposition and Unallocation Routine

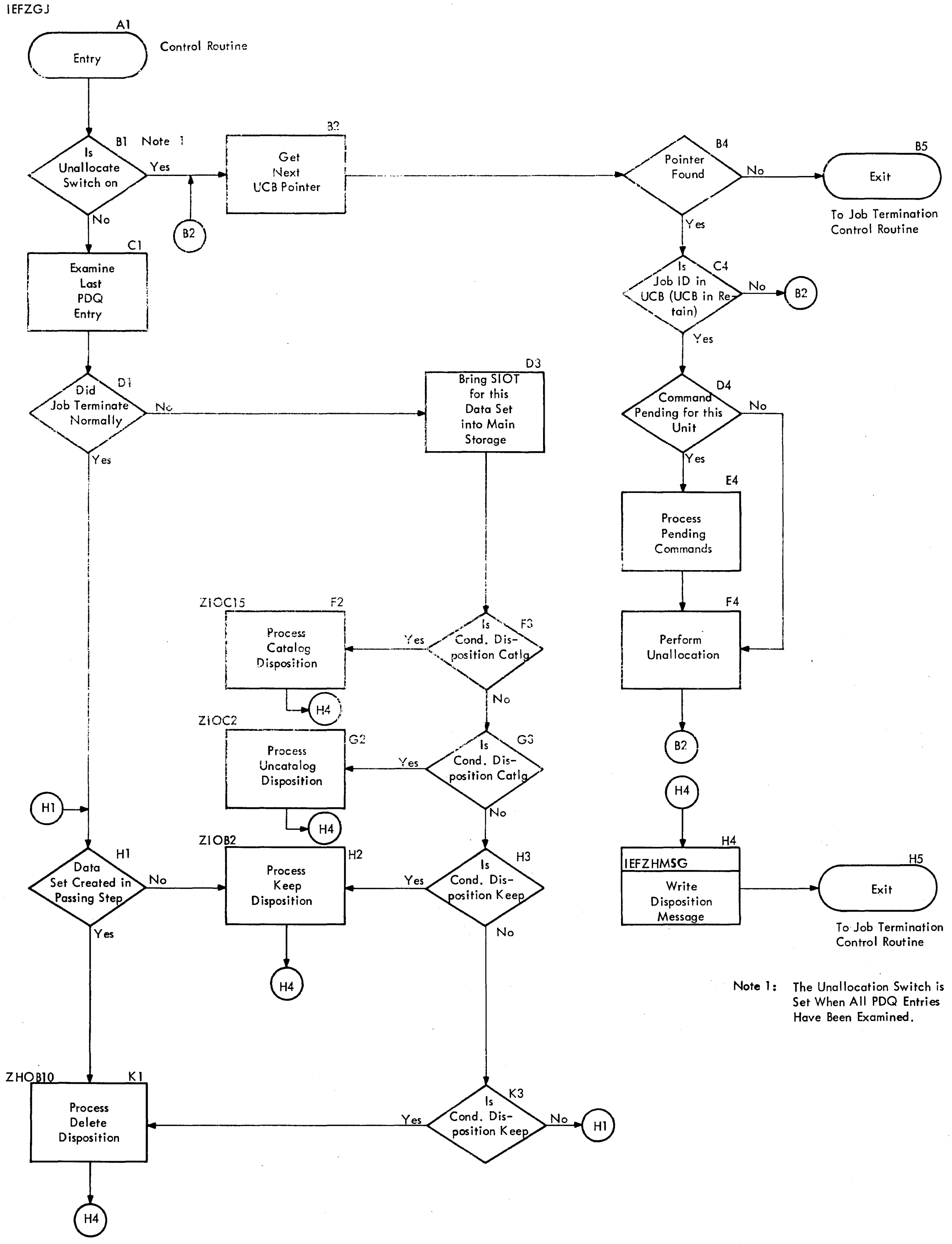

IEFZGJ
A1
Entry
Control Routine
B1
Note 1
Is Unallocate Switch on
Yes
No
B2
B2
Get Next UCB Pointer
B4
Pointer Found
No
B5
Exit
To Job Termination Control Routine
Yes
C1
Examine Last PDQ Entry
C4
Is Job ID in UCB (UCB in Retain)
No
B2
Yes
D1
Did Job Terminate Normally
No
D3
Bring SIOT for this Data Set into Main Storage
D4
Command Pending for this Unit
No
Yes
Yes
E4
Process Pending Commands
ZIOC15
F2
Process Catalog Disposition
Yes
F3
Is Cond. Disposition Catlg
F4
Perform Unallocation
H4
No
ZIOC2
G2
Process Uncatalog Disposition
Yes
G3
Is Cond. Disposition Catlg
B2
H4
H1
H4
No
ZIOB2
H1
Data Set Created in Passing Step
No
H2
Process Keep Disposition
Yes
H3
Is Cond. Disposition Keep
H4
IEFZHMSG
Write Disposition Message
H5
Exit
To Job Termination Control Routine
Yes
No
H4
Note 1: The Unallocation Switch is Set When All PDQ Entries Have Been Examined.
ZHOB10
K1
Process Delete Disposition
Yes
K3
Is Cond. Disposition Keep
No
H1
H4

Chart 32. Queue Management Initialization Routine

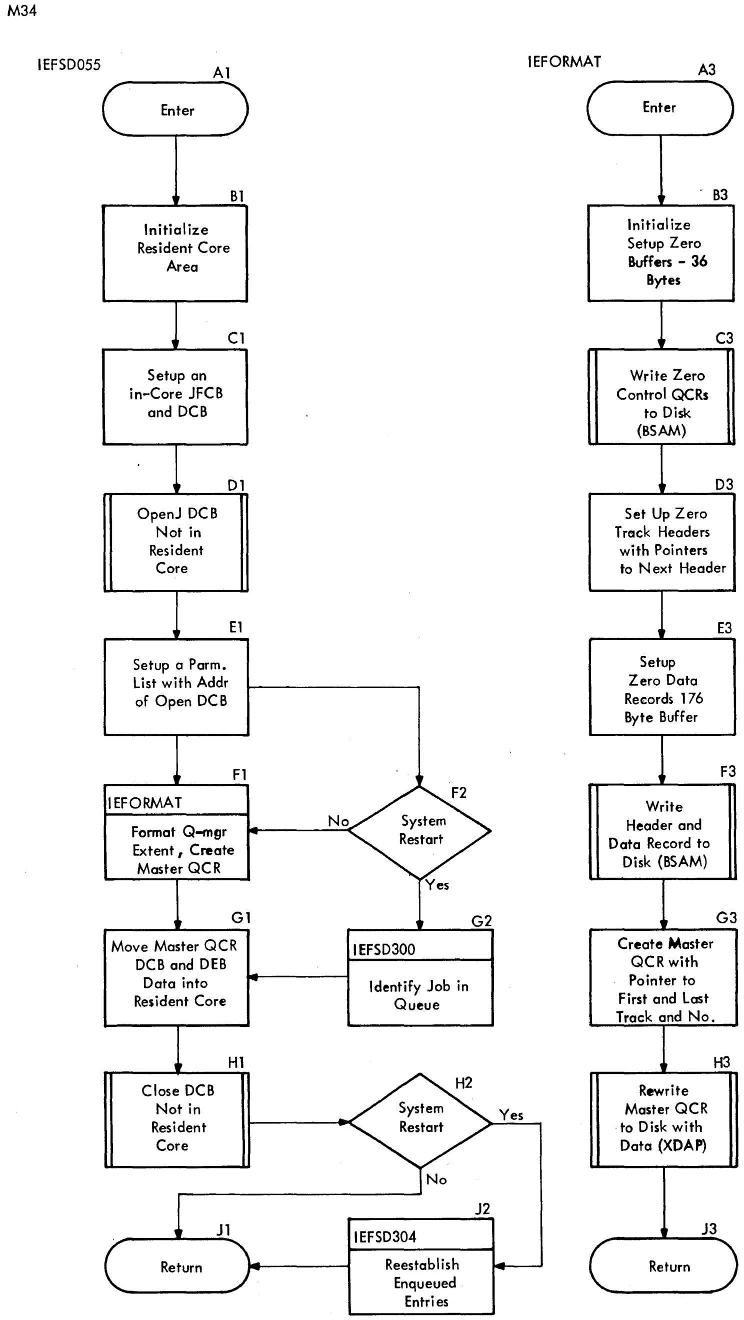

M34
IEFSD055
A1
Enter
B1
Initialize Resident Core Area
C1
Setup an in-Core JFCB and DCB
D1
OpenJ DCB Not in Resident Core
E1
Setup a Parm. List with Addr of Open DCB
F1
IEFORMAT
Format Q-mgr Extent, Create Master QCR
F2
System Restart
No
Yes
G1
Move Master QCR DCB and DEB Data into Resident Core
G2
IEFSD300
Identify Job in Queue
H1
Close DCB Not in Resident Core
H2
System Restart
Yes
No
J1
Return
J2
IEFSD304
Reestablish Enqueued Entries
IEFORMAT
A3
Enter
B3
Initialize Setup Zero Buffers - 36 Bytes
C3
Write Zero Control QCRs to Disk (BSAM)
D3
Set Up Zero Track Headers with Pointers to Next Header
E3
Setup Zero Data Records 176 Byte Buffer
F3
Write Header and Data Record to Disk (BSAM)
G3
Create Master QCR with Pointer to First and Last Track and No.
H3
Rewrite Master QCR to Disk with Data (XDAP)
J3
Return

Chart 33. Queue Management Assign Start and Assign Routine

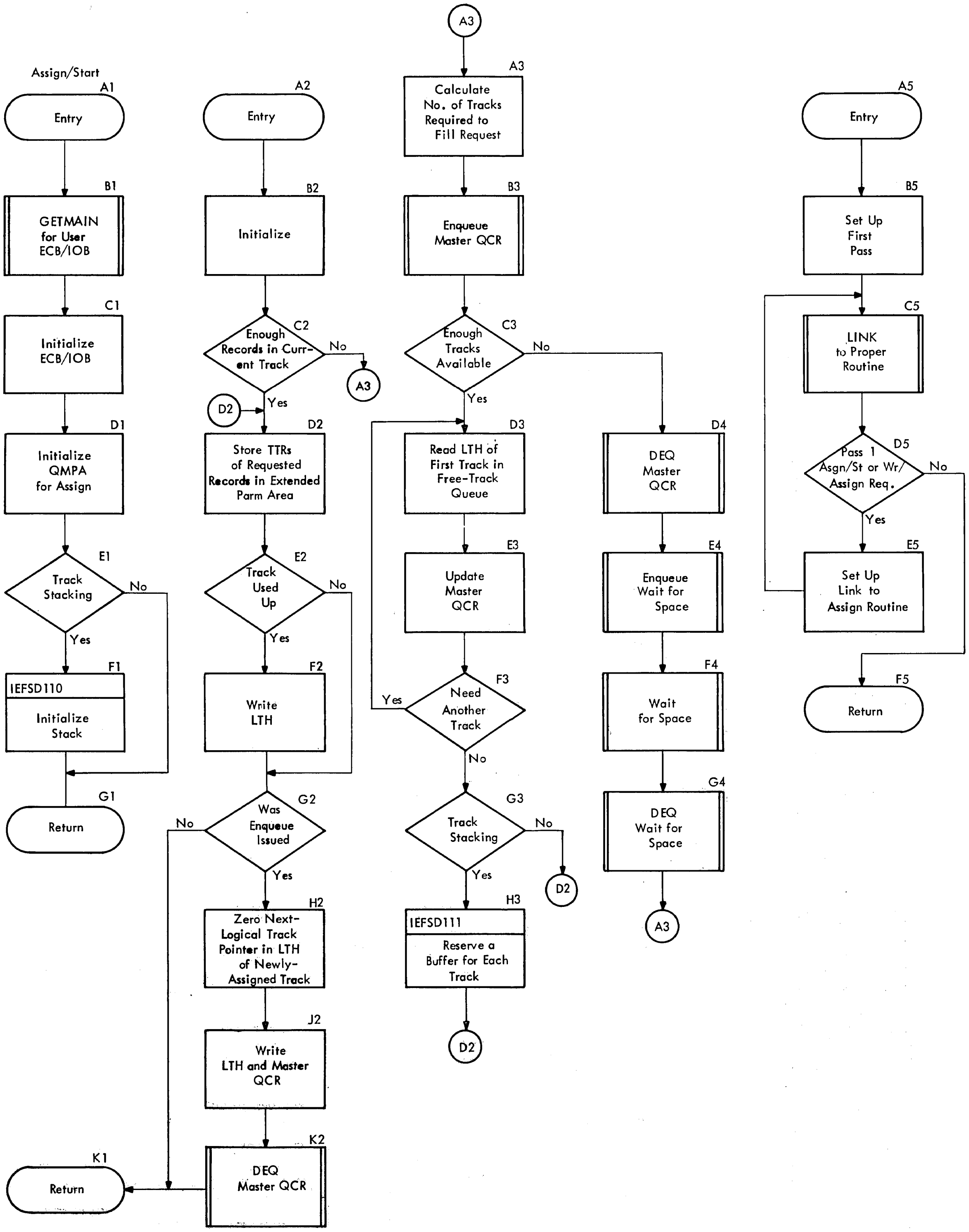


Chart 34. Queue Management Enqueue Routine

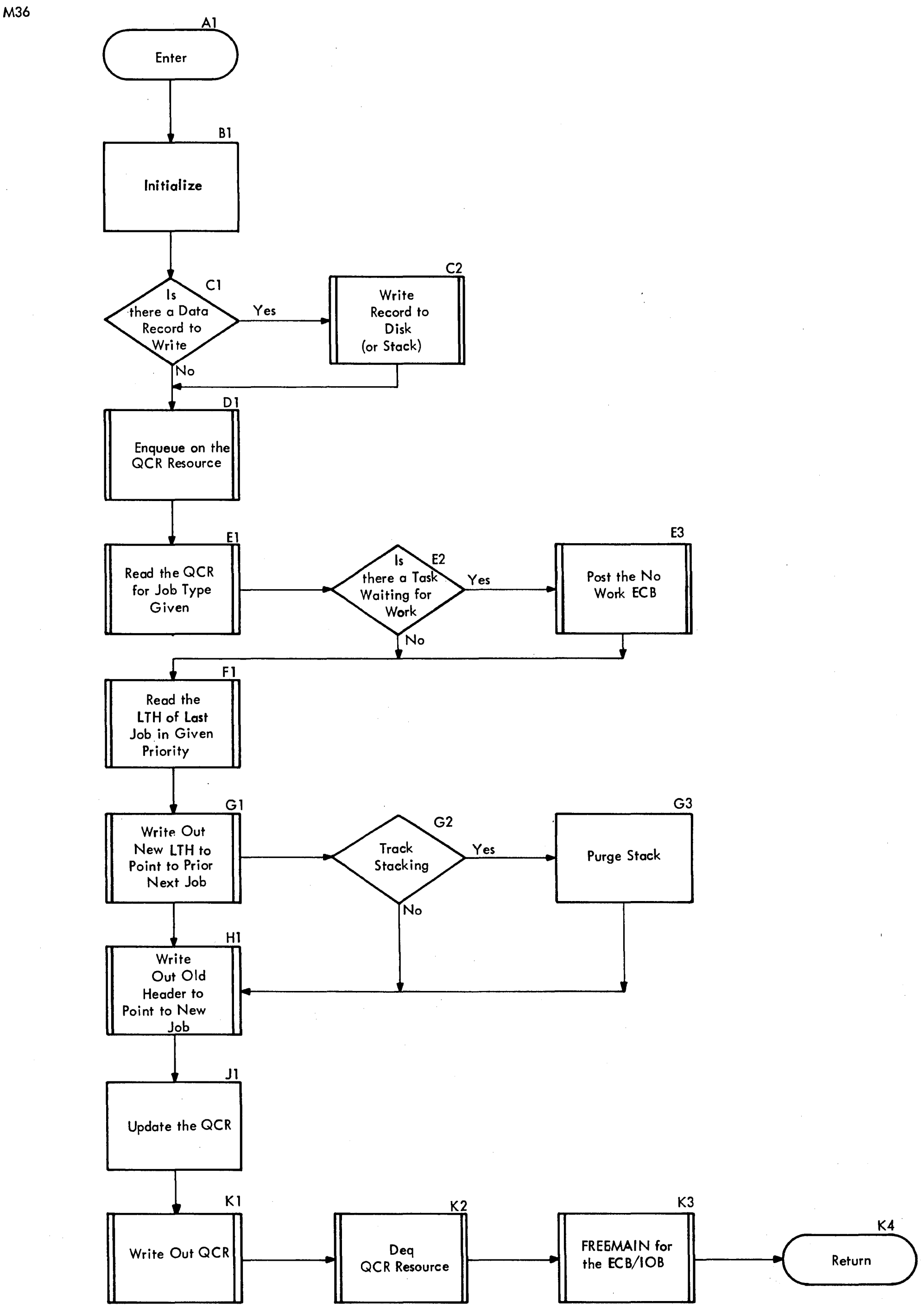

M36
A1
Enter
B1
Initialize
C1
Is there a Data Record to Write
Yes
No
C2
Write Record to Disk (or Stack)
D1
Enqueue on the QCR Resource
E1
Read the QCR for Job Type Given
E2
Is there a Task Waiting for Work
Yes
No
E3
Post the No Work ECB
F1
Read the LTH of Last Job in Given Priority
G1
Write Out New LTH to Point to Prior Next Job
G2
Track Stacking
Yes
No
G3
Purge Stack
H1
Write Out Old Header to Point to New Job
J1
Update the QCR
K1
Write Out QCR
K2
Deq QCR Resource
K3
FREEMAIN for the ECB/IOB
K4
Return

Chart 35. Queue Management Dequeue Routine

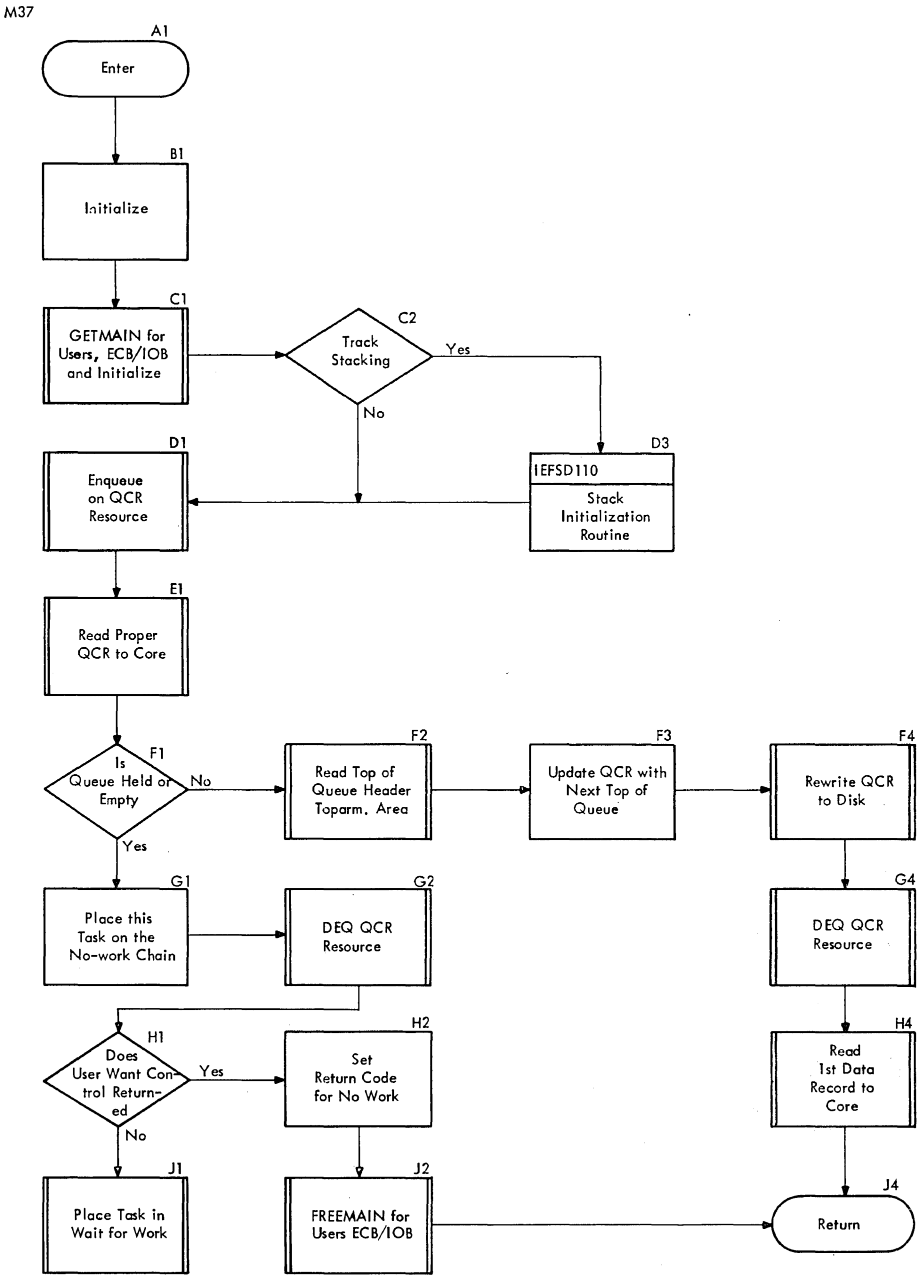
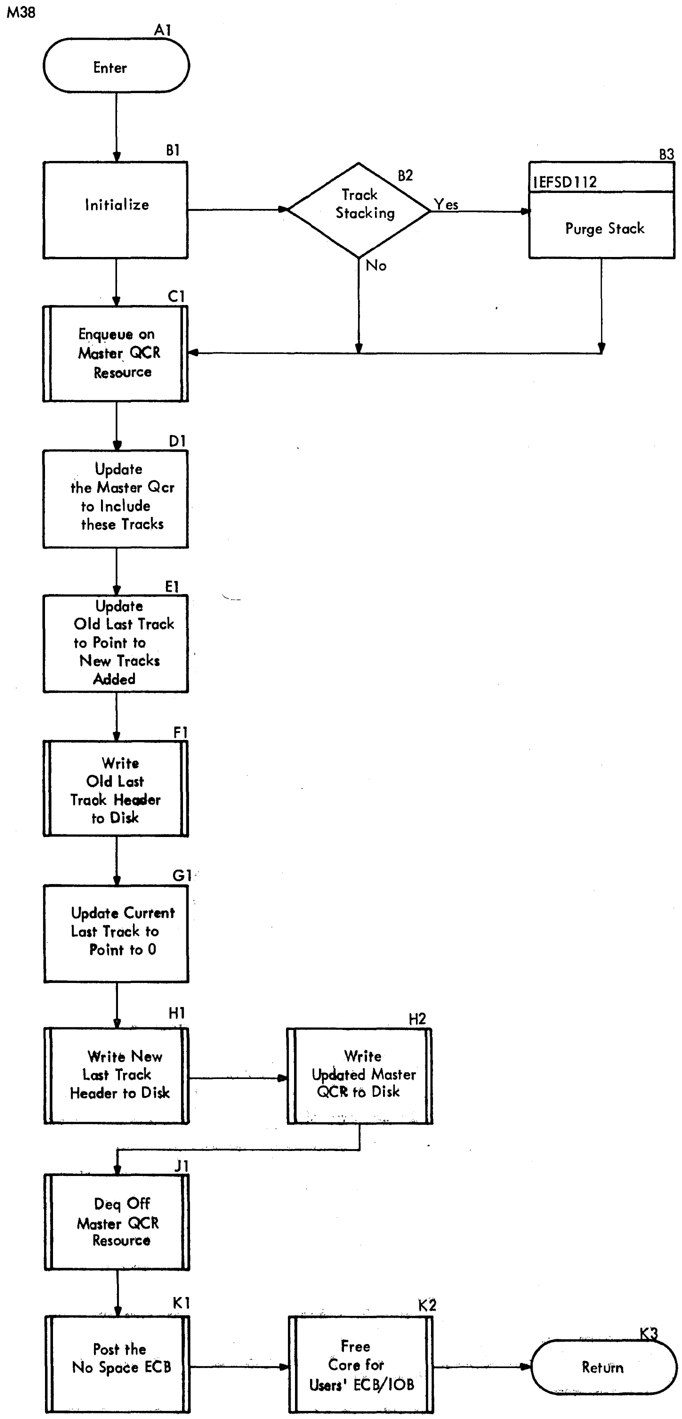

M37
A1
Enter
B1
Initialize
C1
GETMAIN for Users, ECB/IOB and Initialize
C2
Track Stacking
Yes
No
D1
Enqueue on QCR Resource
D3
IEFSD110
Stack Initialization Routine
E1
Read Proper QCR to Core
F1
Is Queue Held or Empty
No
Yes
F2
Read Top of Queue Header Toparm. Area
F3
Update QCR with Next Top of Queue
F4
Rewrite QCR to Disk
G1
Place this Task on the No-work Chain
G2
DEQ QCR Resource
G4
DEQ QCR Resource
H1
Does User Want Control Returned
Yes
No
H2
Set Return Code for No Work
H4
Read 1st Data Record to Core
J1
Place Task in Wait for Work
J2
FREEMAIN for Users ECB/IOB
J4
Return

Chart 36. Queue Management Delete Routine



M38
A1
Enter
B1
Initialize
B2
Track Stacking
Yes
No
B3
IEFSD112
Purge Stack
C1
Enqueue on Master QCR Resource
D1
Update the Master Qcr to Include these Tracks
E1
Update Old Last Track to Point to New Tracks Added
F1
Write Old Last Track Header to Disk
G1
Update Current Last Track to Point to 0
H1
Write New Last Track Header to Disk
H2
Write Updated Master QCR to Disk
J1
Deq Off Master QCR Resource
K1
Post the No Space ECB
K2
Free Core for Users' ECB/IOB
K3
Return

Appendix E: Dictionary of Abbreviations

The following abbreviations are used in the text of this manual.

Abbreviation	Meaning
ACT	account control table
ANSI	American National Standards Institute
APR	alternate path retry
ASB	automatic SYSIN batching
ASBWA	automatic SYSIN batching work area
ASCII	American National Standard Code for Information Interchange
ATCA	allocation/termination communication area
AVR	automatic volume recognition
AVT	allocate volume table
AWA	allocation work area
AWT	allocate work table
BCT	bias count table
BLP	bypass label processing
BRDR	background reader
BSAM	basic sequential access method
CCH	channel-check handler
CCW	channel command word
CIB	command input buffer
CIR	core image record
CLT	channel load table
CPA	communications parameter area
CPU	central processing unit
CRT	cathode ray tube
CSCB	command scheduling control block
CVT	communications vector table
DADSM	direct access device space management
DAR	damage assessment routines
DCB	data control block
DCM	display control module
DDR	dynamic device reconfiguration
DEB	data extent block
DIDOCS	device independent display operator console support
DMT	device mask table
DNT	device name table
DSB	data set block
DSCB	data set control block
DSDR	data set descriptor record
DSENQ	data set enqueue table
DSNT	data set name table
DSO	direct system output
DSOCB	direct system output control block
EBCDIC	extended binary coded decimal interchange code
ECB	event control block
EOV	end of volume
FBQE	free block queue element
FQE	free queue element
GCB	group control block
GDG	generation data group
GWT	GETPART work table
HIR	hardware instruction retry
IOB	input/output block
IPL	initial program load
IWA	interpreter work area

JCL	job control language
JCLS	job control language set
JCT	job control table
JFCB	job file control block
JMR	job management record
JSCB	job step control block
JSEL	job scheduling entrance list
JSOL	job scheduling option list
JSWA	job scheduling work area
JSXL	job scheduling exit list
KBT	keyword branch table
LCE	LOGON communication element
LCT	linkage control table
LTH	logical track header
LWA	local work area
MCH	machine-check handler
MCS	multiple console support
MRCT	message routing control table
M/S	master scheduler
MSB	machine status block
MVCA	mount verification communication area
MVCAX	mount verification communication area extension
MVT	multiprogramming with a variable number of tasks
NEL	interpreter entrance list
NIP	nucleus initialization program
OLTEP	online test executive program
PDQ	passed data set queue
PDT	parameter descriptor table
PFK	programmed function keyboard
PQE	partition queue element
PSA	prefixed storage area
PSW	program status word
QCR	queue control record
QMPA	queue management parameter area
RB	request block
RCB	region control block (TSO)
RJE	remote job entry
RMS	recovery management support
RPS	rotational positional sensing
RQE	reply queue element
SACB	screen area control block
SAMWA	special access method work area
SCD	system output class directory
SCT	step control table
SDT	START descriptor table
SER	system environment recording
SIOP	SIOT processing table
SIOT	step input/output table
SLOT	scheduler lookup table
SMB	system message block
SMF	system management facilities
SPQE	subpool queue element
SQA	system queue area
STAE	set task asynchronous exit
SVC	supervisor call
SVRB	supervisor request block
SYSIN	system input
SYSOUT	system output
TCB	task control block
TCT	timing control table
TIOT	task input/output table
TJB	terminal job block (TSO)
TOD	time-of-day
TQE	timer queue element
TSCVT	time sharing CVT (TSO)

TSO	time sharing option
UCB	unit control block
UCM	unit control module
UCME	unit control module entry
UCMI	unit control module indicator
UCS	universal character set
VOLT	volume table
VQE	vary queue element
VS	variable-length spanned
VTOC	volume table of contents
WQE	work queue element
WTP	write-to-programmer
WTPCB	write-to-programmer control block
XSA	extended save area

Appendix F: Module Cross-Reference

This appendix contains a list of the descriptive names for job management modules. Each name is followed by the module's name as it appears in an assembly listing of the program to which it applies, and as it appears in Appendix B of this publication. Text illustration identifications in which the module name appears are also given. In some figures, modules are shown by their load module names...these names are shown in parenthesis.

Module Descriptive Name	Module Name in Listing	Illustrations (F=Figure, C=Chart)
Allocate/IEFVPOST Communications Block	IEFATECB	
Allocation Control Routine -- I/O Device Allocation	IEFXCSSS	F81
Allocation Exit Routine -- I/O Device Allocation	IEFSD41Q	F81
Allocation Recovery Messages -- I/O Device Allocation	IEFXJMSG	
Allocation Recovery Routine -- I/O Device Allocation	IEFXJIMP	F81
ASB Queue Reader -- Automatic SYSIN Batching	IEFVMG	F7
Assign/Start Routine -- MVT Queue Management	IEFQAGST	
Assign Routine -- MVT Queue Management	IEFQASGQ	
Attach Routine -- MVT Initiator	IEFSD263	F12
Automatic SYSIN Batching -- Command Processor	IEFVMC	
Automatic SYSIN Batching -- Find	IEFVMH	
Automatic SYSIN Batching -- Initialization	IEFVMA	F7
Automatic SYSIN Batching -- Input Stream Processor	IEFVMB	F7
Automatic SYSIN Batching -- Termination	IEFVMD	F7
AVR Allocation Routine -- I/O Device Allocation	IEFXV001	F81
AVR Label Processing Routine -- I/O Device Allocation	IEFXV002	F81,C25
AVR Nonstandard Label Processing Routine -- I/O Device Allocation	IEFXVNSL	
AVR Routine Messages -- I/O Device Allocation	IEFXVMSG	
AVR Unmounted Tape Allocation Routine -- I/O Device Allocation	IEFXV003	
Branch Routine -- MVT Queue Management	IEFQMLK1	C8
CANCEL Processor -- SVC 34	IEE3703D	F24
Chain Manipulator -- SVC 34	IEE0303D	F24
Class Name Set-up Routine -- System Output Writer	IEFSD081	F20
Command Chaining Access Method -- System Output Writer	IEFSDXYZ	F20
Command Processing Routine -- System Output Writer	IEFSD083	F20
Command Processor -- Automatic SYSIN Batching	IEFVMC	F7
Command Router -- SVC 34	IEE5803D	
Command Statement Routine -- Interpreter	IEFVHM	F68,C4
Command Translator -- SVC 34	IEE5403D	F24
Communications Block -- Allocate/IEFVPOST	IEFATECB	
CONTROL Command Handler (Load 1) -- SVC 34	IGC6703D	F24
CONTROL Command Handler (Load 2) -- SVC 34	IEE6803D	
CONTROL Command Handler (Load 3) -- SVC 34	IEE7703D	
CONTROL Command Handler (Load 4) -- SVC 34	IEE7803D	
CONTROL Command Handler (Load 5) -- SVC 34	IEE6903D	
Control Routine -- MVT Queue Management	IEFQBVMS	
Continuation Statement Routine -- Interpreter	IEFVHC	F66,F68,C4
Convert Routine -- System Task Control	IEFCNVRT	F32
Create CSCB -- Post Master Scheduler	IGC0803D	F24
CSCB Creation Routine -- SVC 34	(IGC0803D) IEE0803D	F24,F45

DADSM Error Recovery Routine -- I/O Device Allocation		IEFXT003	C23
Data Set Delete Routine -- System Output Writer		IEFSD171	F20
Data Set Descriptor Record Processing		IEFDSDRP	
Data Set Writer Attach Routine -- System Output Writer		IEFSD070	F20
DD Statement Processor -- Initiator		IEFVDA	C2,C5
DD* Statement Generator Routine -- Interpreter		IEFVHB	F66,C4
Delete Routine -- MVT Queue Management		IEFQDELQ	
Decision Allocation Routine -- I/O Device Allocation		IEFX5000	F81,C20,C25
Dedication Determination Routine -- MVT I/O Device Allocation		IEFSD180	C9,C10,C13
Delete Routine -- MVT System Restart		IEFSD303	F4
Demand Allocation Routine -- I/O Device Allocation		IEFWA000	F12,F81
Demand Strikeout Routine -- I/O Device Allocation		IEFX300A	C25
Dequeue Interface Routine		IEFVRR1	
Dequeue Routine -- MVT Queue Management		IEFQMDQQ	
Device Strikeout Routine -- I/O Device Allocation		IEFX300A	F81
Dictionary Entry Routine -- Interpreter		IEFVGI	C6
Dictionary Search Routine		IEFVGS	
Direct System Output Control		IEFDSCOP	
Direct System Output Determination		IEFDSOAL	F12,C9,C10
Direct System Output Environment Check		IEFDSOCR	F12
Direct System Output Free Block		IEFDSOFB	
Direct System Output STOP and MODIFY Processing		IEFDSOSM	
Direct System Output Tape-to-Printer or Card-Punch Routine		IEFPRTXX	
Direct System Output Wait Routine		IEFDSOLP	
Direct System Output Writer		IEFDSOWR	F12,C27
Display Active		IEEVDSP1	
DISPLAY Active Message Processor -- SVC 110		IEE50110	
DISPLAY C,K Routine (Load 1) -- SVC 110		IEE10110	
DISPLAY C,K Routine (Load 2) -- SVC 110		IEE11110	
DISPLAY C,K Routine (Load 3) -- SVC 110		IEE12110	
Display Configuration Matrix Routine (Load 1) -- SVC 34		IGC6103D	F24
Display Configuration Matrix Routine (Load 2) -- SVC 34		IGC6203D	
DISPLAY CONSOLES Get Region Routine		IEEPDISC	
DISPLAY CONSOLES Processor		IEEXEDNA	
DISPLAY M Routine (Load 1) -- SVC 110		IEE30110	
DISPLAY M Routine (Load 2) -- SVC 110		IEE31110	
DISPLAY M Routine (Load 3) -- SVC 110		IEE32110	
DISPLAY/MONITOR Commands Router Routine -- SVC 34	(IGC3503D)	IEE3503D	F24
DISPLAY PFK Routine -- SVC 110		IEE40110	
Display Region Size		IEEVDRGN	
Display Requests Routine -- SVC 34	(IGC2903D)	IEE2903D	F24
DISPLAY Router		IGC3503D	F24
DISPLAY SQA Routine -- SVC 34		IEE8503D	F24
DISPLAY U Routine (Load 1) -- SVC 110		IEE20110	
DISPLAY U Routine (Load 2) -- SVC 110		IEE21110	
DISPLAY U Routine (Load 3) -- SVC 110		IEE22110	
DISPLAY U Routine (Load 4) -- SVC 110		IEE23110	
Disposition and Unallocation Routine -- Termination		IEFZGJB1	C27,C30
Disposition and Unallocation Messages -- Termination		IEFZGMSG	
Disposition and Unallocation Message Routine -- Termination		IEFZHMSG	C27,C28,C31
Disposition Routine -- Termination		IEFZGST1	C27,C29
DSB Handler Routine -- System Output Writer		IEFSD085	F20
DSB Processing Routine		IEFYTVMS	C27
Dummy Accounting -- Interpreter		IEFACT	
Dummy Accounting -- Termination		IEFACTK	
Dummy Accounting -- Termination		IEFACTRT	C30
Dummy Job Step Module -- MVT Initiator		IEFSD0XX	
Dummy Module -- MVT Queue Management		IEFQMDUM	

Dummy Output Limit Routine	IEFUSO	
Dummy User Job Initiation Exit Routine	IEFUJI	
Dummy User JCL Validation Exit Routine	IEFUJV	F68
Dummy User Step Initiation Exit Routine	IEFUSI	
Dummy User Time Limit Exit Routine	IEFUTL	
DUMP Command Processor (Master Scheduler)	IEE60110	F24
DUMP Processor	IEE1003D	F24
ECB/IOB Construction Routine -- Queue Alter Routine	IEESD582	F37
End-of-File Routine -- Interpreter	IEFVHAA	F66,C4
Enqueue List Construction -- Initiator (Data Set Integrity)	IEFDSLST	
Enqueue Routine -- MVT Queue Management	IEFQMNQQ	F37
Environment Check Routine -- Direct System Output	IEFDSOCR	
Error Condition Router Routine	IKJNULL	
Error Processing Routine -- Write-to-Programmer	IEFWTP02	
EXEC Statement Condition Code Processor Messages -- I/O Device Allocation	IEFVKMSG	
EXEC Statement Conditional Execution Routine -- I/O Device Allocation	IEFVKIMP	F81
EXEC Statement Processor -- Interpreter	IEFVEA	C2,C5
Extended External Action Routine -- I/O Device Allocation	IEFWEXTA	F81,C24
External Action Messages -- I/O Device Allocation	IEFWD001	
External Action Routine -- I/O Device Allocation	IEFWD000	F12,F81 C22,C25
Find -- Automatic SYSIN Batching	IEFVMH	F7
Free Region Routine	IEEPRTN2	F32
Get Parameter Routine -- Interpreter	IEFVGK	C6
Get Region Routine for Queue Initialization -- MVT Queue Management	IEFQINTZ	F4
Get Routine -- Interpreter (Control Routine)	IEFVHA	F66,F68,C2, C4
HALT and SWITCH Commands Processor Routine -- SVC 34	IEE1403D	F24
Hard Copy Message Issuing Routine -- SVC 34 (IGC4103D)	IEE4103D	F45
Indicative Dump Message Module	IEFIDMPM	
Indicative Dump Routine	IEFIDUMP	
Initialization -- Automatic SYSIN Batching	IEFVMA	
Initialization Open Routine -- Interpreter	IEFVH2	C3
Initialization Routine -- Interpreter	IEFVH1	C2,C3
Initialization Routine -- MVT Initiator	IEFSD160	F12,
Initialization Routine -- MVT System Restart	IEFSD300	F4,C32
Initialization Routine -- System Output Writer	IEFSD080	F20,
Initialization Routine -- Write-to-Programmer	IEFWTP00	
Initiator Control -- System Task Control	IEEVICTL	F32
Initiator Wait Routine -- MVT Initiator	IEFSD105	F12
Input Stream Processor -- Automatic SYSIN Batching	IEFVMB	F4
In-Stream Procedure Decompression Interface Routine	IEFVIND	
In-Stream Procedure Directory Build Routine	IEFVINC	
In-Stream Procedure Directory Search Routine	IEFVINB	
In-Stream Procedure Routing Routine	IEFVINA	F68,C2,C4
In-Stream Procedure Syntax Check Routine	IEFVINE	
Interface Control -- MVT Initiator	IEFIIC	F12
Internal JCL Reader -- System Task Control	IEEVRJCL	F32
Interpreter Control -- System Task Control	IEEVRCTL	F32
Internal JCL Reader	IEEVICLR	
Interpreter Controller -- Automatic SYSIN Batching	IEFVMF	F7
Interpreter Data Set Name Table Construction Routine	IEFVDBSD	
Interpreter Exit -- System Task Control	IEEVRC	F32

Interpreter Post Scan Exit Routine -- System Task Control	IEEPSN	F32
Interpreter Region Regulator -- Automatic SYSIN Batching	IEFVME	F7
I/O Device Allocation Interface Routine -- MVT Initiator	IEFSD162	F12
IPL Routine -- Master Scheduler	IEEVIPL	

JCL Edit Routine -- System Task Control	IEEVJCL	F32
JFCB Housekeeping Control Routine and Allocate Processing Routines -- I/O Device Allocation	IEFVMLS1	F81,C9
JFCB Housekeeping Error Message Processing Routine -- I/O Device Allocation	IEFVMLS6	F81,C9
JFCB Housekeeping Error Messages	IEFVMLS7	
JFCB Housekeeping Fetch DCB Processing Routine -- I/O Device Allocation	IEFVM2LS	C9
JFCB Housekeeping GDG All Processing Routine -- I/O Device Allocation	IEFVM4LS	C9
JFCB Housekeeping GDG Single Processing Routine -- I/O Device Allocation	IEFVM3LS	C9
JFCB Housekeeping Patterning DSCB Routine -- I/O Device Allocation	IEFVM5LS	C9
JFCB Housekeeping Unique Volume ID Routine -- I/O Device Allocation	IEFVM76	
Job and Step Enqueue Housekeeping Routine -- Interpreter	IEFVHHB	C4
Job and Step Enqueue Routine--Interpreter	IEFVHH	F66,F68,C4
Job and Step Enqueue Routing Routine -- Interpreter	IEFVKG	C4
Job Delete Routine -- MVT Initiator	IEFSD166	F12
Job Names Routine -- MVT System Restart	IEFSD302	F4
Job Selection Routine -- MVT Initiator	IEFSD161	F12
Job Separator Routine -- System Output Writer	IEFSD094	F20
Job Statement Condition Code Processor -- Interpreter	IEFJIMP	
Job Statement Condition Code Processor Messages	IEFVJMSG	
Job Statement Processor -- Interpreter	IEFVJA	C2,C5
Job Suspension -- Initiator	IEFSD168	F12
Job Termination Control Routine -- Termination	IEFZASB3	C27
Job Termination Exit Routine -- Termination Routine	IEFSD31Q	C27
Job Validity Check Routine -- Interpreter	IEFVHEC	F66,F68,C4

Link Table Module -- System Task Control	IEEVLNKT	F32
Link to Queue Manager Delete Routine	IEFSD709	F20
Linkage Module	IEF061SD	
Linkage Module	IEF065SD	
Linkage Module	IEF078SD	
Linkage Module	IEF079SD	
Linkage Module -- I/O Device Allocation	IEFWCFAK	
Linkage Module -- I/O Device Allocation	IEFWDFAK	
Linkage Module -- I/O Device Allocation	IEFWSWIN	
Linkage Module -- I/O Device Allocation	IEFXJFAK	
Linkage Module -- MVT	IEF300SD	
Linkage Module -- MVT	IEF304SD	
Linkage Module -- MVT Initiator	IEF060SD	
Linkage Module -- System Output Writer	IEF082SD	
Linkage Module -- System Output Writer	IEF083SD	
Linkage Routine -- Interpreter	IEFIRCB	
Linkage to IEFVMLS1 -- I/O Device Allocation	IEFVMFAK	
Linkage to IEFXJIMP -- I/O Device Allocation	IEFSD551	
Linkage to IEFXJIMP -- I/O Device Allocation	IEFSD552	
Linkor Routine -- System Output Writer	IEFSD078	F20
Locate Log Data Sets	IEEVLIN1	F1

Machine Status Control Routine: Model 85; Part 1 -- SVC 34	IGF08501	F25

Machine Status Control Routine: Model 85; Part 2 -- SVC 34		IGF08502	F25
Machine Status Control Routine: Model 145 -- SVC 34		IGF29701	F25
Machine Status Control Routine: Model 155 -- SVC 34		IGF29601	F24
Machine Status Control Routine: Model 165 -- SVC 34		IGF55301	F24
Main Logic Routine -- System Output Writer		IEFSD082	F20
Master Scheduler Get Region		IEEPPRES	F1
Master Scheduler IPL Routine		IEEVIPL	F1
Master Scheduler Public/Private Interface		IEEVPRES	F1
Master Scheduler Resident Data Area		IEEBASEA	
Master Scheduler Wait and Attach Routine		IEEVWAIT	F1
MCS Console Information (Load 2) -- SVC 34		IEE7303D	
MCS Reply Processor Routine -- SVC 34		IEE1A03D	
MCS/TSO Periodic STOP Command Processor		IEE5503D	F24
MCS VARY Syntax Check -- SVC 34	(IGC3303D)	IEE3303D	F45
Message Assembly -- SVC 34	(IGC0503D)	IEE0503D	F32,F37,F45
Message Assembly Routine -- SVC 34		IEE2103D	
Message Blocking Routine		IEFYSVMS	C27
Message Module -- Command Scheduling Routine		IGG2103D	F45
Message Module for Warmstart -- Termination		IEFWSMSG	
Message Module (Headers and Tests for Messages)		IEFSD096	F81
Message Module Indicative Dump		IEFIDMPM	
Message Module -- I/O Device Allocation		IEFWSTRT	
Message Module -- I/O Device Allocation		IEFXAMSG	
Message Module (Messages 00-07) -- Interpreter		IEFVGM1	
Message Module (Messages 08-0D) -- Interpreter		IEFVGM78	
Message Module (Messages 08-0F) -- Interpreter		IEFVGM2	
Message Module (Messages 10-17) -- Interpreter		IEFVGM3	
Message Module (Messages 18-1F) -- Interpreter		IEFVGM4	
Message Module (Messages 20-27) -- Interpreter		IEFVGM5	
Message Module (Messages 28-27) -- Interpreter		IEFVGM6	
Message Module (Messages 30-3F) -- Interpreter		IEFVGM7	
Message Module (Messages 38-3F) -- Interpreter		IEFVGM70	
Message Module (Messages 3E-45) -- Interpreter		IEFVGM19	
Message Module (Messages 50-57) -- Interpreter		IEFVGM8	
Message Module (Messages 56-5D) -- Interpreter		IEFVGM17	
Message Module (Messages 58-5F) -- Interpreter		IEFVGM9	
Message Module (Messages 60-67) -- Interpreter		IEFVGM10	
Message Module (Messages 68-6F) -- Interpreter		IEFVGM11	
Message Module (Messages 70-77) -- Interpreter		IEFVGM12	
Messages Module (Messages 78-7F) -- Interpreter		IEFVGM13	
Message Module (Messages 80-87) -- Interpreter		IEFVGM18	
Message Module (Messages 88-8F) -- Interpreter		IEFVGM14	
Message Module (Messages 90-97) -- Interpreter		IEFVGM15	
Message Module (Messages A0-A7) -- Interpreter		IEFVGM16	
Message Module -- MVT Queue Management		IEFSD311	
Message Module -- MVT System Restart		IEFSD312	
Message Module -- SVC 34		IEE7903D	F24
Message Module -- Termination		IEFWTERM	
Message Module -- Volume Attribute		IEFK1MSG	F1
Message Processing Routine -- Interpreter		IEFVGM	F68,C6
Message Processing Routine -- Write-to-Programmer		IEFWTP01	
Message Routine -- Queue Alter Routine		IEESD580	F37
Message Writing Routine -- System Task Control		IEEVSMSG	F32
MODE Command Router Routine		IEF2603D	F24
MONITOR Router		IGC7103D	F24
MOUNT Command Syntax Check		IEEVMNT1	
MOUNT Command Control		IEEVMNT2	
MOUNT Control Volume Routine -- I/O Device Allocation		IEFMCVOL	C9,C13,C15, C16,C17,C22
M/S Router -- SVC 110		IEE00110	
MSGRT Processor		IGC6303D	F24
MSGRT Command Routine (Load 2) -- SVC 34		IEE6403D	
MSGRT and CONTROL (MR and K) Message Module (Load 1) -- SVC 34		IEE5603D	

MSGRT and CONTROL Message Module (Load 2) -- SVC 34		IEE5903D	
MVT Queue Management Branch Routine		IEFQMLK1	
MVT Queue Management Dummy Module		IEFQMDUM	
MVT Allocation Entry Routine -- MVT I/O Device Allocation		IEFSD21Q	F81
MVT Interpreter Dummy Module		IEFKGFAK	
MVT Linkage to JFCB Houskeeping		IEFVMMS1	
M65MP Vary Preprocessor	(IGC3603D)	IEE3603D	F45
Non-recovery Error Routine -- I/O Device Allocation		IEFXKIMP	F81
Non-recovery Error Routine Messages -- I/O Device Allocation		IEFXKMSG	
NULL Statement Routine -- Interpreter		IEFVHL	F66,C4
Open Routine for SVC 83		IEESMFOP	
Operator Message Routine -- Interpreter		IEFVHR	
Periodic STOP Command Handler -- SVC 34		IEE4503D	F24
Post-Attach Interface Routine -- MVT Initiator		IEFSD104	F12
Post-Scan Routine -- Interpreter		IEFVHF	C4,C5
Pre-Attach Routine -- MVT Initiator		IEFSD103	F12
Pre-Scan Preparation Routine -- Interpreter		IEFVHEB	F68,C4
Print Line Routine -- System Output Writer		IEFSD095	F20
Printer Routine -- System Output Writer		IEFSDTTE	
Program Properties Table Module -- MVT Initiator		IEFSDPPT	F12
Public/Private Interface -- Master Scheduler		IEEVPRES	
Purge Queue Construction Routine -- MVT System Restart		IEFSD301	F4
PUT Routine -- System Output Writer		IEFSD089	F20
Queue Alter Delete Routine -- Queue Alter Routine		IEESD576	F37
Queue Alter Get Region Routine		IEEPALTR	F37
Queue Alter Syntax Check Routine		IEESD562	F37
Queue Formatting Routine -- MVT Queue Management		IEFORMAT	F4,C32
Queue Initialization Routine -- MVT Queue Management		IEFSD055	F4,C32
Queue Management Convert Routine		IEFCVNRT	
Queue Management Dequeue by Jobname Routine		IEFLOCDQ	
Queue Management Error Return Routine		IEFXTDMY	
Queue Management Interface Routine -- Interpreter		IEFVHQ	
Queue Restart Enqueue Routine -- Queue Alter Routine		IEESD577	F37
Queue Restart Message Class Set-up Routine -- Queue Alter Routine		IEESD578	F37
Queue Scratch Routine -- Queue Alter Routine		IEESD581	F37
Queue Scratch Setup Routine -- Queue Alter Routine		IEESD575	F37
Queue Search Operand Processor		IEESD584	
Queue Search Return -- Queue Alter Routine		IEESD583	F37
Queue SMB Routine -- Queue Alter Routine		IEESD579	F37
Reader Control Routine -- Interpreter		IEFIRC	
Read/Write Routine -- MVT Queue Management		IEFQMRAW	
Read/Write Routine -- System Task Control		IEFRDWRT	F32
Record Accessing Program -- MVT Queue Management (Track Stacking)		IEFSD111	C33
Record Decompression Routine -- Interpreter		IEZDCODE	
Record Decompression Routine -- Interpreter		IEZNCODE	
Record Transfer -- SVC 83		IEESMF8C	
Reenqueue Routine -- MVT System Restart		IEFSD305	F4
Reinterpretation Control Routine		IEFVRRC	
Reinterpretation Delete/Enqueue Routine		IEFVRR3	
Replace Region Interface Routine -- MVT Initiator		IEFSD101	F12
Replace Region Routine -- MVT Initiator		IEFSD102	F12
Reply Message Routine -- SVC 34		IEE1B03D	
Reply Processor -- SVC 34		IEE1203D	

Resident Main Storage Reservation Routine -- MVT Queue Management		IEFQRESD	
Resident Wait Routine		IEELWAIT	F1
Restart Activation Routine		IEFVSDRA	F4,F12
Restart Determination Routine		IEFVSDRD	F4
Restart Preparation Routine		IEFRPREP	F4,C27
Restart Reader Linkage Routine		IEFRCLN1	F10
Restart Reader Linkage Routine		IEFRCLN2	F10
Restart SVC Issuing Routine		IEFRSTRT	F10
RJE Read/Write Routine -- Queue Management		IEFRDWRT	
RJF Write LTH Routine		IEFSD447	
Router -- Interpreter (Control Routine)		IEFVHE	F66,F68,C4
Router Routine -- SVC 34		IEE0403D	F24
Routing Location Routine (Load 1) -- SVC 34		IEE7503D	F24
Routing Location Routine (Load 2) -- SVC 34		IEE7603D	F24

Scan Routine		IEFVSCAN	
Scan Routine -- Interpreter		IEFVFA	C2
SCD Construction Routine -- MVT Interpreter		IEFVSD13	
Scratch Data Sets Routine -- MVT System Restart		IEFSD304	F4,C32
Scratch Data Sets Routine -- MVT System Restart		IEFSD308	F4
Search Routine -- Queue Alter Routine		IEESD564	F37
Search Control Routine -- Queue Alter Routine		IEESD563	F37
Separation Strikeout Routine -- I/O Device Allocation		IEHXH000	F81,C20
Service Routine -- Queue Alter Routine		IEESD565	F37
SET Command Handler	(IGC0903D)	IEE0903D	F24
SET Command Part I -- SVC 34	(IGC0603D)	IEE0603D	F24
SET Command Part II -- SVC 34	(IGC8603D)	IEE8603D	F24
SET Time-of-Day (TOD) Clock -- SVC 34	(IGC6503D)	IEE6503D	F24
SIOT Reader for Warmstart -- Termination		IEFWSYP3	C27
SMB Handler -- System Output Writer		IEFSD086	F20
SMB Reader Routine		IEFVSMBR	F10
SMF Allocation Routine -- SVC 83		IEESMFAL	
SMF Dump Routine		IFASMFDP	
SMF Initialization		IEESMFIT	F1
SMF Initialization		IEESMFI2	F1
SMF Open Initializer		IEESMFOI	F1
SMF Parameter Processor		IEESMFI3	F1
SMF VARY Record Handler Routine -- SVC 34	(IGC2303D)	IEE2303D	F45
SMF Writer		IEESMFWR	F1
SMF Writer Interface Routine -- Termination		IEESMFWI	
Space Request Routine -- I/O Device Allocation		IEFXT00D	F81,C23
Spanned Data Sets Handler -- System Output Writer		IEFSDXXX	F20
Spool Routine -- Interpreter		IEFVHG	C4
Stack Initialization -- MVT Queue Management (Track Stacking)		IEFSD110	C33,C35
Stack Purge Routine -- MVT Queue Management (Track Stacking)		IEFSD112	C36
STAE Environment Creation Routine -- SVC 34		IEE0003D	F24
STAE Exit Routine (Load 1) -- SVC 34		IEE5103D	F24
STAE Exit Routine (Load 2) -- SVC 34		IEE5203D	F24
STAE Exit Routine (Load 3) -- SVC 34		IEE5303D	F24
Standard Writer Routine -- System Output Writer		IEESD087	F20
START and MOUNT Commands Get Region Routine		IEEPRWI2	F32
START Command Syntax Check -- System Task Control		IEEVSTAR	F32
START/MOUNT hierarchy Parameter Processor -- SVC 34	(IGC3803D)	IEE3803D	F24
Step Termination -- Termination		IEFYNMSG	
Step Termination Control Routine -- Termination		IEFYNIMP	C27
Step Termination Data Set Driver Routine -- Termination		IEFYPJB3	C27
Step Termination Exit Routine -- Termination		IEFSD22Q	C27
Step Termination Messages -- Termination		IEFYPMSG	
STOP and MODIFY Processing Routine -- Direct System Output		IEFDSOSM	
STOP and MODIFY Scheduling Routine -- SVC 34		IEE0703D	F24

Symbolic Parameter Processing Routine -- Interpreter		IEFVFB	C2
System Output Interface Routine -- MVT Termination		IEFSD017	
SYSIN Processor Routine -- MVT Interpreter		IEFVSD12	
Table Merge Routine		IEFVRR2	
Table Breakup Routine -- System Restart		IEFSD514	F4,F12
Table Build Routine Translate Table -- Data Set Integrity		IEFDSTRT	
Tape-to-Printer or Card-Punch Routine -- Direct System Output		IEFPRTXX	
Task Delete Routine -- MVT Initiator		IEFSD164	F12
TCTIOT Construction Routine -- Initiator		IEFSMFAT	
Termination Entry Routine -- Termination Routine		IEFSD42Q	
Termination Routine -- MVT Interpreter		IEFVHN	F66,C2,C4
Test and Store Routine -- Interpreter		IEFVGT	C6
TIOT Compression Routine -- I/O Device Allocation		IEFXT002	C23
TIOT Construction Routine -- I/O Device Allocation		IEFWCIMP	F81
TOD Clock, TQE Update -- SVC 34	(IGC6603D)	IEE6603D	F24
Transient Queue Management Initialization and Read/Write Routine		IEFXQM00	
Transient Queue Management Record Assignment Routine		IEFXQM02	
Transient Queue Management Track Assignment Routine		IEFXQM01	
Transition Routine -- System Output Writer		IEFSD088	F20,
Tree Table Build Routine -- Initiator (Data Set Integrity)		IEFDSTBL	
TTR and NN to MBBCCHHR Conversion Routine -- MVT System Restart		IEFSD310	
Unallocation Routine -- Termination		IEFZGST2	C27,C28,C29
Unchain Routine -- MVT Queue Management		IEFGMUNQ	
Unsolicited Device Interrupt Handler -- MVT I/O Device Allocation		IEFVPOST	
User Accounting Routine Linkage -- Termination		IEFACTLK	
User Exit Initialization Routine -- Initiator		IEFSMFIE	
User Exit Initialization Routine -- Termination Routine		IEFSMFLK	
VARY and UNLOAD Processor -- SVC 34	(IGC3103D)	IEE3103D	F24,F45
VARY and UNLOAD Syntax Scan -- SVC 34	(IGC1103D)	IEE1103D	F24,F45
VARY Channel Routine		IEEMPVCH	
VARY/CONSOLE/HARDCPY Keyword Scan -- SVC 34	(IGC4403D)	IEE4403D	F45
VARY CONSOLE Information Message -- SVC 34	(IGC4803D)	IEE4803D	F45
VARY CONSOLE Processor -- SVC 34	(IGC4903D)	IEE4903D	F45
VARY CPU Routine		IEEMPVCP	
VARY HARDCOPY UNIT Processor -- SVC 34		IEE7203D	
VARY HARDCPY, OFF Routine -- SVC 34	(IGC5703D)	IEE5703D	F45
VARY HARDCPY Processor -- SVC 34	(IGC4703D)	IEE4703D	F45
VARY Keyword Router -- SVC 34	(IGC3203D)	IEE3203D	F24,F45
VARY MSTCONS Processor -- SVC 34	(IGC4303D)	IEE4303D	F45
VARY ONLINE/OFFLINE for Console Devices -- SVC 34	(IGC4603D)	IEE4603D	F45
VARY Operand Check Routine	(IGC2203D)	IEEMPCKR	F45
VARY PATH Processor for Multiprocessing Routine (Part 1) -- SVC 34	(IGC2403D)	IGF24MPD	F45
VARY PATH Processor for Multiprocessing Routine (Part 2) -- SVC 34		IGF34MPD	
VARY PATH Processor Routine -- SVC 34		IGF2403D	
Vary Preprocessor -- M65MP	(IGC3603D)	IEE3603D	F45
VARY Secondary Keyword Scan -- SVC 34	(IGC4203D)	IEE4203D	F45
VARY Storage Routine		IEEMPVSE	
Verb Identification Routine		IEEVHCB	F66,F68,C4
Volume Attribute Setting Routine		IEFPRES	F1

Wait and Attach Routine (Master Scheduler)	IEEVWAIT	
Wait for Unallocation Routine -- MVT Device Allocation	IEFSD195	F81
Wait for Space Decision Routine -- MVT I/O Device Allocation	IEFSD097	F81
Wait Routine -- Direct System Output	IEFDSOLP	
Wait Routine -- System Output Writer	IEFSD084	F20
Write JFCBs	IEELOG01	F1
3211 Printer Processor	IEFSDTTE	F20

Bibliography

The following list of books provide additional background information for understanding the material contained in this publication:

IBM System/360 Operating System:

System Control Blocks, GC28-6628

Operator's Reference, GC28-6691

Operator's Procedures, GC28-6692

Introduction, GC28-6534

Data Management Services, GC26-3746

Supervisor Services and Macro Instructions, GC28-6646

Data Management Macro Instructions, GC26-3794

MVT Guide, GY28-6720

This publication also makes reference to the following publications:

IBM System/360 Operating System:

Graphic Job Processor Support PLM, GY27-7159

Direct Access Device Space Management PLM, GY28-6607.

Input/Output Supervisor PLM, GY28-6616.

MVT Supervisor PLM, GY28-6659.

IPL/NIP PLM, GY28-6661.

Remote Job Entry PLM, GY30-2005.

System Management Facilities, GC28-6712.

TSO Command Processor PLMs, GY28-6771 through GY28-6777 [7 volumes]

TSO Control Program PLM, GY28-7199.

Component Summary -- 3830 Storage Control; 3330 Disk Storage, GA26-1592.

Index

Indexes to program logic manuals are consolidated in the publication IBM System/360 Operating System: Program Logic Manual Master Index, GY28-6717. For additional information about any subject listed below, refer to other publications listed for the same subject in the Master Index.

READER'S COMMENT FORM

IBM System/360 Operating System:
MVT Job Management
Program Logic Manual

Order No. GY28-6660-9

Please use this form to express your opinion of this publication. We are interested in your comments about its technical accuracy, organization, and completeness. All suggestions and comments become the property of IBM.

Please do not use this form to request technical information or additional copies of publications. All such requests should be directed to your IBM representative or to the IBM Branch Office serving your locality.

- Please indicate your occupation: ______________________________

- How did you use this publication?
  - □ Frequently for reference in my work.
  - □ As an introduction to the subject.
  - □ As a textbook in a course.
  - □ For specific information on one or two subjects.

- Comments (Please include page numbers and give examples.):

- Thank you for your comments. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS, PLEASE . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Note: Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Cut Along Line

Fold Fold

FIRST CLASS
PERMIT NO. 81
POUGHKEEPSIE, N.Y.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY ...

IBM Corporation
P.O. Box 390
Poughkeepsie, N.Y. 12602

Attention: Programming Systems Publications
Department D58

Fold Fold

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]

System/360 OS MVT Job Management PLM (S360-36) Printed in U.S.A. GY28-6660-9

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]